Math

Report No. 382

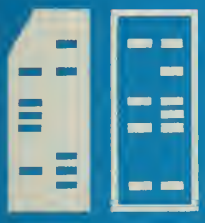
PARALLEL SIMULATION OF DIGITAL SYSTEMS

by

Chiyozi Tanaka

April 30, 1970

ILLIAC IV Document No. 211



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 382

PARALLEL SIMULATION OF DIGITAL SYSTEMS*

by

Chiyozi Tanaka

April 30, 1970

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. USAF 30(602)-4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, January 1970.

ABSTRACT

The paper describes the method used to generate the parallel simulator developed for the ILLIAC IV Project, University of Illinois. The basic approach to achieve an efficient simulator system is parallel logic simulation, package-level simulation, the use of a high level language (ALGOL) and a flexible simulation control language (TESLA). This paper analyzes the problems associated with the ideas and gives the algorithms used to generate the simulator. Emphasis is placed on the algorithms and the implementation of the simulator generator system.

ACKNOWLEDGMENT

The author would like to express his deep appreciation to Professor Kenji Naemura for his guidance and many helpful suggestions in the writing of this thesis. Also, many thanks to Mr. Arthur B. Carroll and to Mr. Luther C. Abel for their advice and criticisms. The author is also indebted to his co-worker, Prathima A. Agrawal, who wrote part of the programs of the system.

Thanks are also extended to the Diagnostic Group personnel for their helpful discussions.

Furthermore, many thanks go to Mrs. Kay Flessner, who did such a fine job of typing this thesis.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. CONSIDERATION OF A SIMULATOR DESIGN 4

3. ORGANIZATION 10

3.1 Simulator Generator Group 10

3.2 Simulator 11

3.3 Result Display Group 12

3.4 Simulator Control Language 12

4. ALGORITHMS FOR THE SIMULATOR BODY GENERATOR 17

4.1 Definitions 17

4.2 Level Assignment 19

4.2.1 Level Assignment for a Link Graph 19

4.2.2 Determination of All Nodes Bounded by Maximal Strongly Connected Subgraphs 24

4.2.3 Level Assignment for Any Graph 24

4.3 Loop Detection 26

4.3.1 Determination of R Matrix 26

4.3.2 Determination of Maximal Strongly Connected Subgraphs 28

4.4 Ordering 33

4.4.1 Determination of the Longest Paths in a Graph [6] . . . 36

4.4.2 An Algorithm to Find the Longest Paths 39

4.5 Algorithm for Generating the Simulator Body 41

Page

5. SIMULATION CONTROL LANGUAGE (TESLA) 48

5.1 Declaration 48

5.1.1 Program 48

5.1.2 Declarations 48

5.2 Control Statement 50

5.2.1 Input Control Statements 50

5.2.2 Storage Element Control Statements 50

5.2.3 Output Control Statements 51

5.3 A Sample Program 52

6. SIMULATOR CONTROLLER 56

7. IMPLEMENTATION 58

7.1 TESLA Compiler 58

7.2 Simulator Generator Group 58

7.3 Simulator 62

7.4 Result Display Program 62

8. CONCLUSION 64

APPENDIX . . . . . .

A A Sample Update Listing of CUBC/UPDATE 66

B A Sample Signal Sort Listing of CUBD/WEDTR1 67

C Listings of CUBD/WEDTR2 69

D Listings of CUBD/LEVELER 74

E Listings of CUBD/REDUCE 78

F Listings of CUBD/HSKEEP 80

G Listings of CUBD/SIMGEN 84

H A Sample Output Listing of <BOARD NAME>/SIMULA 86

LIST OF REFERENCES 91

LIST OF FIGURES

Figure		Page
1.	General Flow of Logic Simulator System	14
2.	Flow of Simulator Generator System	15
3.	The Function of Simulator	16
4.	An Example of Ascending Level Assignment	22
5.	Level Assignment of Any Directed Graph	25
6.	An Example of the Determination of Maximal Strongly Connected Graphs in G	29
7.	An Example of Node Ordering and its Logical Connections	34
8.	The Modified Order and Its Logical Connections	35
9.	An Example of the Determination of the Longest Paths	38
10.	An Example of a Graph for the Simulator Body Generator	43
11.	Extracted Graph Bounded by Loops	44
12.	Condensed Graph of the Example	46
13.	An Example of TESLA Program	54
14.	A Sample Format of TESLA Object Command	57
15.	Program System of Simulator	63

1. INTRODUCTION

The introduction of integrated circuitry and large scale integration is changing the trend in logical design and simulation of computers [4]. Electrical and mechanical limitations on semiconductor and packaging technology place significant restrictions on logic design and the partitioning and logic design must be an iterative process to arrive at a completed design.

Because of the nature of the fabrication, the logic of a machine must be described in several levels such as a) the logic in an IC package, b) the interconnection of IC's within a printed circuit board (on which 20 ~ 150 IC packages can be mounted in general), c) the connection among printed circuit boards within one section, d) the connection among sections.

From the view of the manufacturing processes, the logic in an IC package or printed circuit wiring on a printed circuit board cannot be changed as easily as the discreetly wired connection within a section; therefore, debugging an original design before the design is committed to hardware is being increasingly emphasized. Additionally, as the design of computers has become more complicated and more sophisticated, it has become almost impossible for a designer to design an error-free computer without the assistance of some tool to examine his design.

One of the purposes of the logic simulator system is to provide the designer with more immediate design help by which he can exercise ample test programs on his logic and obtain detailed information to determine if his design is correct. After the logic design is debugged, the machine will be constructed and checkout of the machine follows. In this stage, failure detection or isolation of faulty components is required.

The simulator can be used not only for debugging the logic of a computer, but also for the generation of diagnostic test patterns, i.e., if the simulator is modified so that a description of failures of each element is incorporated in the simulator, it can calculate the output pattern for a certain failure (fault insertion) and certain input pattern; and if this output pattern is different from the correct output pattern, which is calculated by the original (correct) simulator, then the input pattern which produced these outputs, can detect the above failure. Thus, if the simulator is modified so as to simulate failures in the logic, the simulator can be used to generate fault detection tests [9]. In order to generate fault test patterns, the simulator must basically be executed for each failure by applying input patterns until some input pattern detects a given failure. In general as the input patterns are randomly chosen, the simulator must be executed many times to find the patterns for all possible failures. For this purpose, the speed of execution of the simulator is very important.

Many efforts in the field of logic simulation have already been made. The most frequently used technique for a logical simulation is a so-called gate-level simulation, which means that a set of logical equations is compiled into a string of executable computer instructions and the simulation is accomplished by executing the resulting program a given number of clock times. However, the disadvantage of the gate level logic simulator is that regardless of mechanical organization of the logic, the logical equations of the whole machine to be simulated must be specified in one single description to the simulator; therefore, if a machine is too big to be simulated at one time, the logical designer must partition the whole system into several blocks

by hand and write all the equations for each block, which in turn causes extra work and less accuracy of logic debugging. In this paper, a simulator system is presented which is generated from the interconnection list (wire list) and preserves the mechanical organization of the logic.

Although it is necessary to have several simulators of various levels, such as a printed circuit board simulator, a section simulator and a machine simulator, only the printed circuit board simulator is described, which is used extensively for ILLIAC IV printed circuit board debugging and test pattern generation, because the other simulators are designed in the same way.

2. CONSIDERATION OF A SIMULATOR DESIGN

The requirements for a simulator should vary according to the organization and the fabrication of a computer. Desirable specifications for a logic simulator including fault test generation can be described as follows:

1) The construction of the simulator should conform as closely as possible to mechanical organization of the logic.

The logical description of a computer will be partitioned to IC package, printed circuit board, section and system, and the only available information from the design process will be the logic description of IC packages, the connection list within a printed circuit board, the connection list within one section and the connection list among sections within a system. Therefore, the logic equations of a whole system will not necessarily be produced for the production process.

From this point of view, it is desirable to design a logic simulator which requires as input a description of the IC packages used in the design and the wire-list. In addition, once a basic simulator - a printed circuit board simulator - is designed, it is desirable to construct the other simulators using the same procedures as the basic simulator by just collecting each printed circuit board simulators.

2) The simulator should be easily updated.

The logic internal to an IC package may not be changed as easily as the connections on a board (during the design phase) or the backplane connections among boards. The construction of the logic simulator should have the same nature. It is desirable that any change in design may be reflected in different ways in the simulator; therefore, the simulator can be much more easily updated if those different connections can be distinguished one from the others.

3) The logic described in the simulator should be easily understandable to a designer.

In general logic debugging may be done by reviewing logical equations and/or logic diagram. If the simulator gives a designer a complete set of logical equations, the designer can directly work with the simulator to find input patterns for the simulator or to debug his logic and to change the logic.

4) Failure insertion to the simulator should be achieved with minimum effort.

After completion of logic debugging, the simulator will be used for generation of failure detection tests. This is achieved by inserting additional information for each element which specifies the failure modes of the elements into the simulator; therefore, this modification should be as easily as possible.

5) The speed of the simulator should be fast.

One of the most important points for a simulator is how fast the simulator is. Since a computer performs Boolean operations on full data words, it is possible to associate a different test case with each bit in one word. Those test cases in one word will then be calculated in parallel; therefore, the effective speed of the simulation will be increased as many times as the number of usable bits of one word.*

In order to accomplish those objectives, four major ideas are employed in the simulation system. They are parallel processing logic simulation, package-level simulation, the use of a high level language (ALGOL) and a flexible simulation control language called TESLA (TEst Simulator LAnguage).

Package level simulation is similar to gate-level simulation. However, input data to the simulator generator is a logic description of each IC package and the connections between those IC packages in the case of a printed circuit board simulator, therefore, the designer need not write all the Boolean equations for input to the simulator. If the wirelist and the logic description of the IC packages are correct, the simulator can be generated automatically. The idea of package-level simulation is that the logic of each IC package is described in one subroutine and the simulator body which means the simulator program without its control program for controlling it or communicating with its user is a well-ordered sequence of subroutine call state-

*The computer used for this simulation system is Burroughs-5500. One word consists of 48 bits and 47 bits are usable. Hence, 47 different test cases can be evaluated in parallel by this system.

ments and the associated parameters correspond to the signals among those IC packages. Hence, only the logic of a package is described by a set of Boolean equations.

This technique can be easily utilized to a section simulator in which one subroutine described the logic on one printed circuit board and the simulator body is again a sequence of subroutine calls and the parameters correspond to the interface signals between those printed circuit boards. Therefore, it is advantageous to use a high level language so that a subroutine can define the connection between packages. Besides, the logic of the simulator is more easily understood when written as a single high-level language statement rather than a series of assembly code statements. To modify the simulator to permit failure insertion, it is then only necessary to modify the sets of logic equations which describe the logic of each IC package type.

This approach, however, gives rise to several problems to be solved as follows:

1) How to order the packages involved in loops.

2) To which level to assign packages containing storage elements.

3) The requirements for parallel processing.

4) How to obtain better efficiency of the simulation.

The subroutine calls must be ordered properly so that the propagation of control and data signals can be followed. The order is defined by

the assigned levels and the simulation of packages is executed in accordance with the levels. However, as the assignment of levels is not originated from the logic equations of packages but from the wiring between packages, there may exist loops for which the level cannot be assigned. This loop means that the input to some package is dependent on the outputs from a previous package whose input is in turn dependent on the outputs of the package before that and so on until the package whose inputs are dependent on the output of the original package, assuming that there is no logical loop but a physical loop. For the first problem, since the level cannot be assigned to those packages, one level is assigned to a set of looped packages and the simulation is performed iteratively for these packages until all of the logical values within the loop have stabilized.

The second problem can be solved by duplicating a storage package into two nodes. Since this simulator evaluates logic at one clock time and the output of a storage element does not change its value during the period and its input value is transferred to its output only at the end of one clock time simulation; hence it can be divided into two parts; the output part having no incoming signal and the input part having no outgoing signal. If a storage package is thus partitioned, the level of the output part is assigned to the first level and the level of the input part to the last level.

For the third problem, each signal occupies one word whose bits correspond to test cases. Binary Boolean operations are executed on full word on the Burroughs-5500, but if the Boolean value of one word is interrogated, only the forty-seventh bit is evaluated; and if this bit is locial one, its entire value is true and otherwise false. Therefore, if tests of entire Boolean values are required, such as the stabilization test, comparison with

each bit in a word should be made. The other case is when a signal is always a constant value, such as a constant voltage connected to a pin. If logical one is assigned to some signals, this corresponding signal should be filled with all ones instead of being assigned with true, in which case only the forty-seventh bit is set to one. Therefore, arithmetic expressions are used instead of Boolean.

In order to obtain better efficiency, several improvements can be considered. As already stated, the package level simulation is based on a sequence of subroutine calls, however, the linkage of those subroutine will take considerable time. Hence, instead of generating the subroutine calls, it is more efficient to generate a sequence of sets of logical equations explicitly, in which case each pin of a package is replaced by a signal name.

The second point of the improvement is that if the loop consists of many packages, the efficiency of the simulation - how fast the values of looped packages will stabilize - depends on the order of the packages within a loop. The efficient iterative order of simulation of the packages is along the longest path within the loop which does not pass through any package twice.

The third improvement is that if there are more than one independent loops within unassigned packages, more efficient simulation can be obtained by assigning each loop to a different level although the original level assignment program may assign those independent loops to the same level in order to minimize the number of packages within a program loop. However, the second and third improvements are not implemented in the current system because the number of looped package is so few that significant advantages cannot be achieved. The level assignment, loop detection and ordering algorithms are described in the following section in detail.

3. ORGANIZATION

The logic simulator system consists of four major groups of programs: the simulator generator group, the simulator itself, the result display group and simulator control language compiler. The general flow of this system is shown in Figure 1. [1]

3.1 Simulator Generator Group

The input to this group is a wirelist which describes package interconnections and descriptions of each IC package used on printed circuit board. First, the wirelist is edited so that it contains only logical signals for the simulation and is updated if necessary, and each signal is classified to input, output and internal signal with respect to the connector of a board so that these signals may belong to proper corresponding arrays in the simulator. Then a representation of the board in the form of a directed graph is produced; in which a node of the graph corresponds to each IC package and an arc of the graph to connections from a package's output to another package's input. Before generating the directed graph, a package of storage element is split into two nodes, one corresponding to the package's output and the other to the package's input.

During these processes signal sort listing and pin sort listing are generated; and if there are errors in the wirelist, error messages are printed for reference to a designer.

The directed graph representation is used for input to the level assignment program. Ascending and descending level assignment are carried out as far as possible. The remaining nodes will be either members of loops or

bounded by loops. For those remaining nodes, loop detection is carried out and each loop is replaced by a pseudo node. Once all loops are replaced by pseudo node, no loop exists in this subgraph of the remaining nodes, therefore, level assignment is again carried out for this subgraph so that all packages and pseudo-nodes are assigned to proper level. Each loop is examined to be ordered for efficient simulation and then the final determination of the order of simulation is made. With this output in which each package is ordered properly for the correct and efficient simulation and signal array list generated by the first part of this group, ALGOL codes are generated to simulate the logical function of each IC package associated with the node.

In case of looped packages, before generating sets of Boolean equations a program loop statements and stability test statements are added. The variables used in these Boolean equations are the simulation variables associated with the signals connected to those packages.

Those sets of logical equations in ALGOL coding called the simulator body is merged with manually prepared simulator control program called "head and tail," and a complete simulator program will be produced automatically. The flow of this group is shown in Figure 2.

3.2 Simulator

The simulator consists of three major sections: the simulator head, the simulator body, and the simulator tail.

The simulator body consists of properly ordered sets of logical equations which describe the function of all packages within a printed circuit board generated by the simulator body generator group whose function is that given inputs to a board and the states of all storage elements on the board

at any particular time, the simulator body will calculate the outputs from the board and the next states of all storage elements.

The simulator head consists of control routines which pass interface signal values and storage element states to the simulator body and translate various control commands generated by the TESLA compiler to control the execution of the simulation with a given set of signal values.

The simulator tail governs the creation of an output file which contains the simulation results and also controls the simulation such as the termination and, in conjunction with the simulator head, controls looping or iteration through the simulator in simulation of sequential clock pulses. The function of the simulator is shown in Figure 3.

3.3 Result Display Group

The result display program processes the simulation result file and decompose this parallel simulation output to the output of each test case. The outputs are then edited and printed in a desired form which is specified by the print commands of the simulator control language.

3.4 Simulator Control Language

To permit a designer to utilize the full range of simulation capabilities, a simple simulator control language (TESLA) is used for various control of the simulation. TESLA is fundamentally an assembler-like language. Statements in the language consist of declarations and test steps. Declarations are used to associate lists of interface and storage signal names with single identifiers (internally array elements), and similarly to associate bit patterns with identifiers.

A Test Step is defined to be the simulation of one clock interval. During a test step the following actions will occur in the order indicated. a) Adjust the input values and storage element contents; b) Simulate the action of the combinational logic during the interval; c) Store the output values and next storage element states. For the control of these actions, a test step with respect to TESLA consists of a step label followed by a series of input, storage element and output control statements. One prominent feature of this language is that a declaration or a statement can specify simultaneous generation and control of many simulations. The further description of the language is described in Section 5.

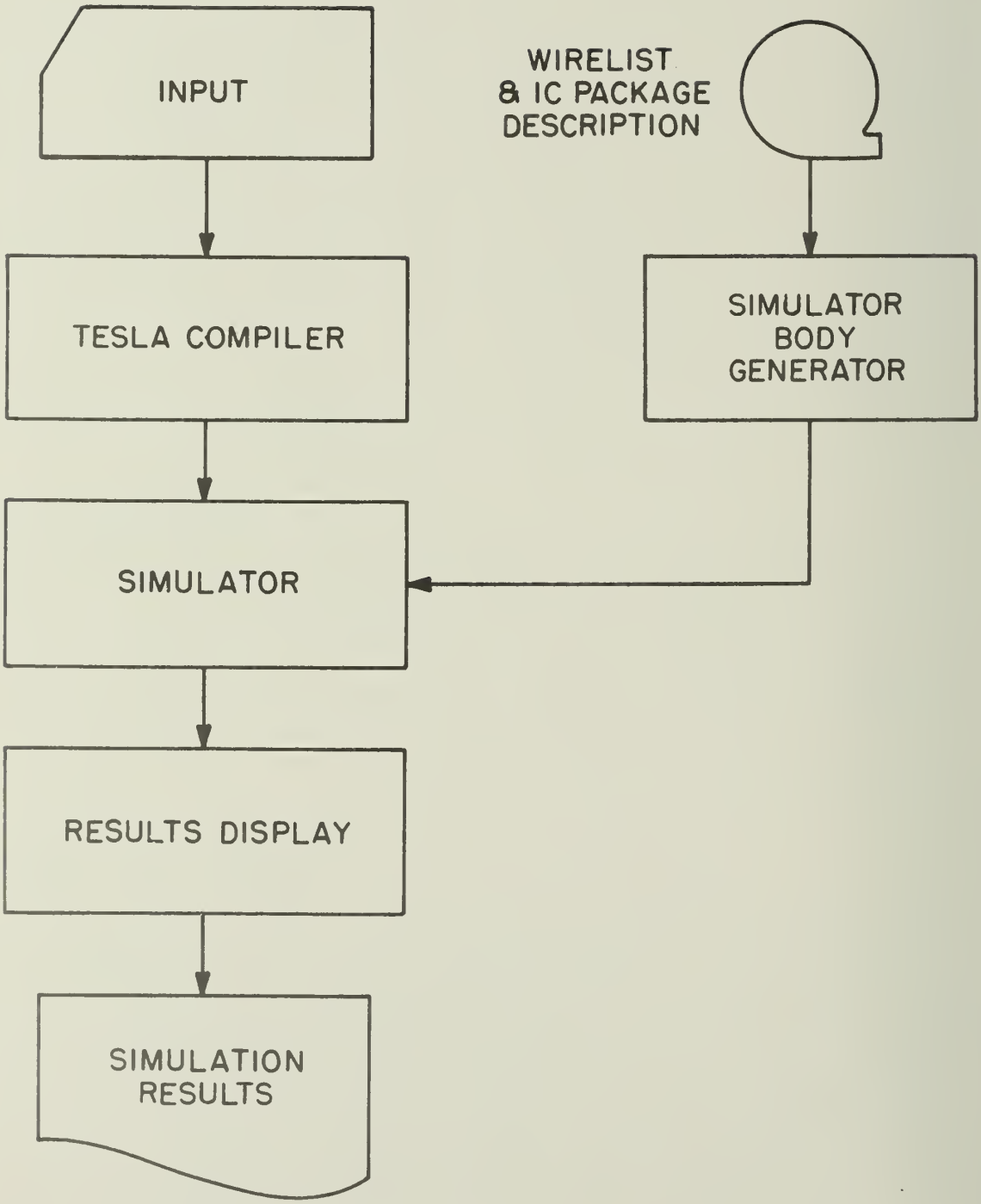


Figure 1. General Flow of Logic Simulator System.

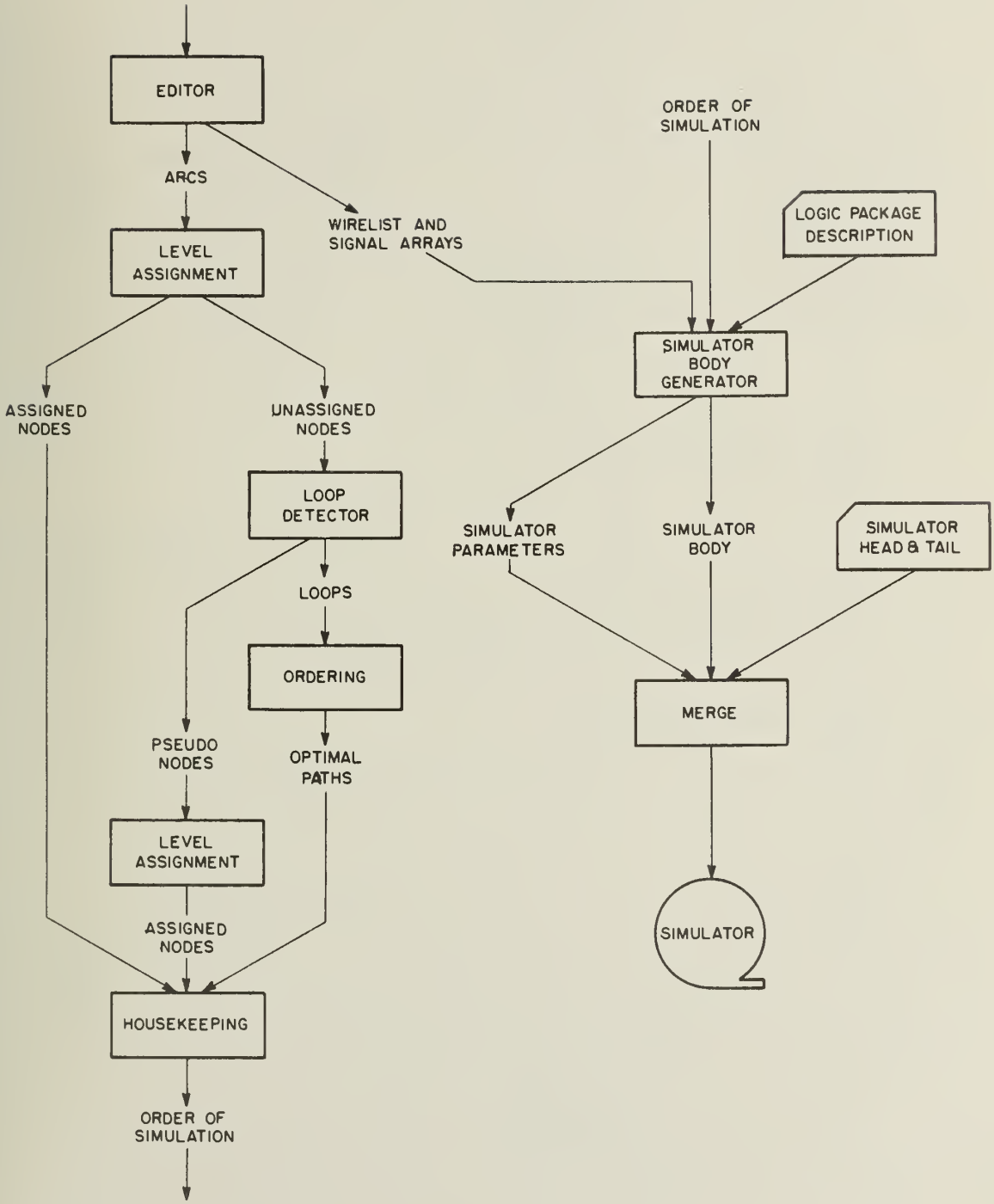


Figure 2. Flow of Simulator Generator System.

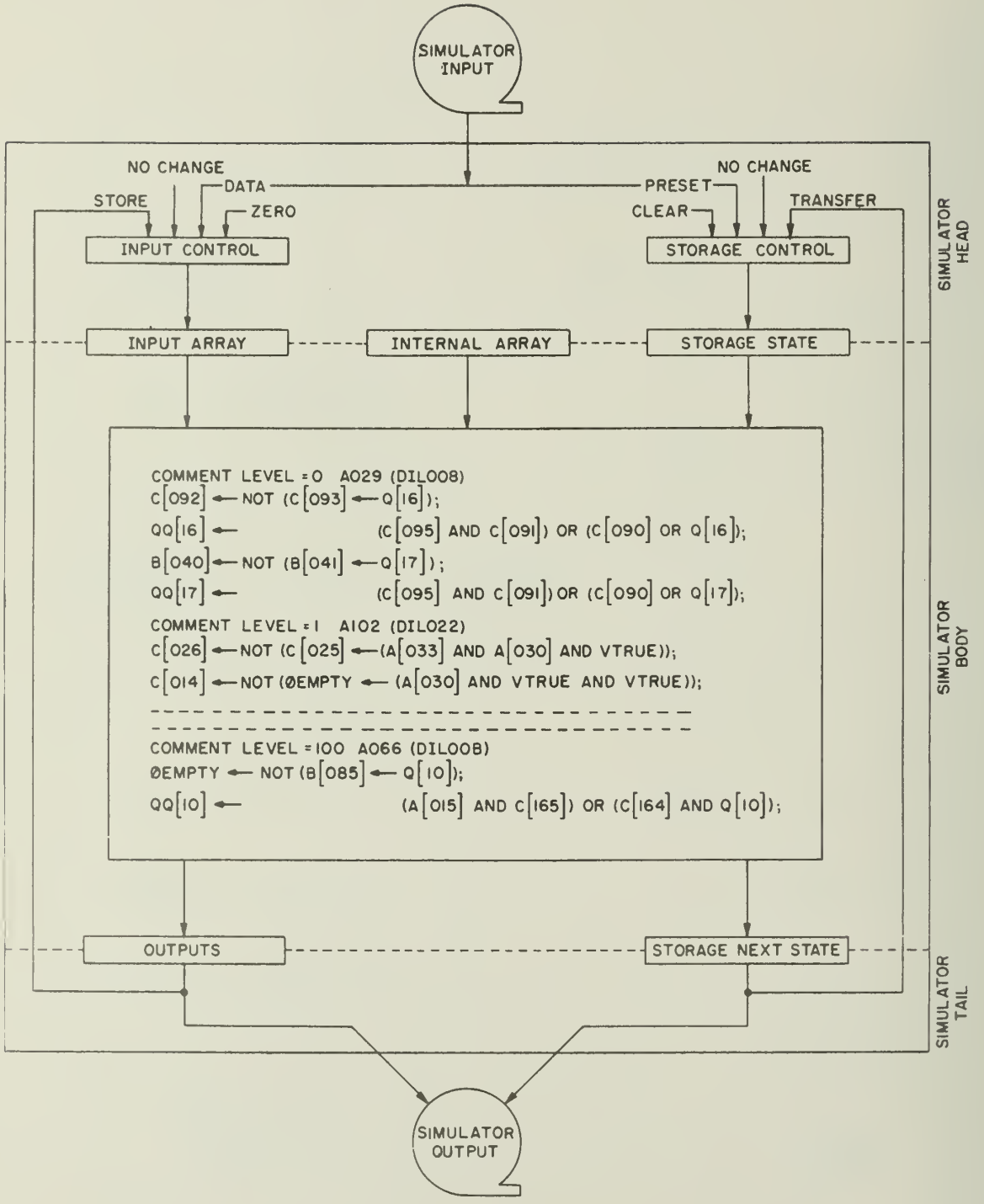


Figure 3. The Function of Simulator.

4. ALGORITHMS FOR THE SIMULATOR BODY GENERATOR

The problems of level assignment, loop detection and the ordering can be clearly defined in term of directed graph [3], [5], [7], [8]. In order to convert from a wirelist to the graph, the wirelist must be changed to a directed branch list in which a node in the graph corresponds to a package and a directed branch to a connection between packages so that there is no parallel lines between any two nodes since parallel lines cannot be distinguished from each other in a graph.

The following definitions are necessary to establish those algorithms:

4.1 Definitions

A directed graph is a set of nodes connected by directed branches. A path is a sequence of directed branches connected in series between two nodes without passing a node twice. A loop is a sequence of nodes such that a path exists from a node to itself. Node j is said to be reachable from node i if there is at least one directed path from node i to node j. A subgraph of a given graph is defined as a subset of nodes with all branches between these nodes retained. A graph consisting of a set of nodes and branches is said to be strongly connected if and only if any nodes are reachable from any others. The maximal strongly connected subgraph is a strongly connected subgraph which includes all possible nodes which are strongly connected with each other. A link graph is a graph which has no loops. Mathematical definition of a graph can be stated as follows:

A graph, which is denoted by $G = (X, \Gamma)$ is the pair consisting of the set X of nodes and the function Γ on X. If x and y are two nodes such that $Y \in \Gamma x$, they are joined by a branch pointing from x to y. The following operations of Γ are defined:

i) $\Gamma^2 x = \Gamma (\Gamma x)$

$\Gamma^n x = \Gamma (\Gamma^{n-1} x) = \overbrace{\Gamma \Gamma \ldots\ldots \Gamma}^{n} x$

ii) The transitive closure of Γ is a function $\hat{\Gamma}$ defined by

$\hat{\Gamma} x = \{ x \} \cup \{\Gamma x \} \cup \{\Gamma^2 x \} \cup \{\Gamma^3 x \} \cup \ldots.$

$\hat{\Gamma}^n x = \{ x \} \cup \{\Gamma x \} \cup \{\Gamma^2 x \} \cup \ldots\ldots \cup \{\Gamma^n x \}$

iii) The inverse of Γ is a function defined by

$\Gamma^{-1} y = \{ x \mid y \in \Gamma x \}$

If B is a subset of X in a given graph $G = (X, \Gamma)$

$\Gamma^{-1} B = \{ x \mid \Gamma x \cap B \neq \phi \}$

A graph can also be represented by a square matrix of order n, where n is the number of nodes

iv) An adjacency matrix A(G) of a graph G is defined by

$$A(G) = [a_{ij}]$$

$$a_{ij} = \begin{cases} 1 & \text{if } x_j \in \Gamma x_i \\ 0 & \text{if } x_j \notin \Gamma x_i \end{cases}$$

v) A reachability matrix R(G) of a graph G which indicates whether node x_j is reachable from node x_i is defined by

$$R(G) = [\gamma_{ij}]$$

$$\gamma_{ij} = \begin{cases} 1 & \text{if } x_j \in \hat{\Gamma} x_i \\ 0 & \text{if } x_j \notin \hat{\Gamma} x_i \end{cases}$$

4.2 Level Assignment

4.2.1 Level Assignment for a Link Graph

The level assignment is to assign integer values to the nodes of a link graph G. The graph G has an ascending level assignment, and the integers are levels if for each branch (a_i, a_j), $(a_j \in \Gamma a_i)$ in G the corresponding integers satisfy $n_i < n_j$. Also the graph G has a descending level assignment, if for each branch (a_i, a_j), $(a_j \in \Gamma a_i)$ in G the corresponding integers satisfy $n_i > n_j$.

The assignment of levels to a given link graph is defined by the following:

Let X be a set of all nodes in a link graph G.

Let a_i (i = 1, 2, ... p) be the node of the graph G.

a) Ascending level assignment

i) The set A(0) of level zero nodes is defined by

$$A(0) = \{ a_i \mid \Gamma^{-1} a_i = \Phi \}$$

ii) The set A(n) of level n nodes is defined by

$$A(n) = \{ a_i \mid \Gamma^{-1} a_i \subset \bigcup_{j=0}^{n-1} A(j) \}$$

b) Descending level assignment

i) The set D(0) of level zero nodes is defined by

$$D(0) = \{ a_i \mid \Gamma a_i = \Phi \}$$

ii) The set D(n) of level n nodes is defined by

$$D(n) = \{ a_i \mid \Gamma a_i \subset \bigcup_{j=0}^{n-1} D(j) \}$$

From these definitions the following theorem results:

Theorem 1. The following properties of a directed graph G are equivalent.

(1) G has no strongly connected subgraph, i.e., a link graph.

(2) It is possible to order the nodes of G so that its adjacency matrix is upper triangular.

(3) It is possible to assign levels n_i to the node a_i in such a way that if $a_i \in \Gamma a_j$ in G, then $n_i > n_j$. (This assignment is of ascending order and the same argument is also true for descending level assignment.)

Proof:

(1) → (2)

Since G has no strongly connected subgraph, there exists at least one node which has no incoming branches. Let those nodes be A(0).

If B is a subset of the nodes of G, we define G-B as the subgraph obtained by deleting the nodes of B and all branches of the graph originating in B. Since G has no strongly connected subgraph, G - A(0) also has no strongly connected subgraph, therefore G - A(0) also has at least one node which has no incoming branches. Let those nodes be A(1). By continuing the process, all nodes of G can be partitioned to one of A(i). With the nodes of G ordered in this way, let $A(G) = [a_{ij}]$ be its adjacency matrix. Since by the construction of this matrix, there is no branch from a node of A(i) to a node of A(j) in G if $i \geq j$. Hence the corresponding entry $a_{ij} = 0$.

(2) → (3)

Suppose the nodes of A(G) have been ordered so that A(G) is upper triangular. A(i) can be assigned to level i since $a_{ij} = o$ if $i > j$, i.e., there is no branch from a node of one level to one of a lower level.

(3) → (1)

If G satisfies (3), then all nodes in G must be assigned levels. However, suppose there is a strongly connected subgraph in G and let the member of the node be

$$a_1, a_2, \cdots a_n. \quad (n \geq 1)$$

Then $a_i \in \hat{\Gamma} a_k$ and $a_k \in \hat{\Gamma} a_i$ $\quad (i, k=1, 2, \ldots n)$

and $n_i > n_k$ and $n_i < n_k$ which contradicts (3)

therefore G has no strongly connected subgraph.

This completes the proof of theorem 1.

Example 1. Ascending level assignment of Figure 4.

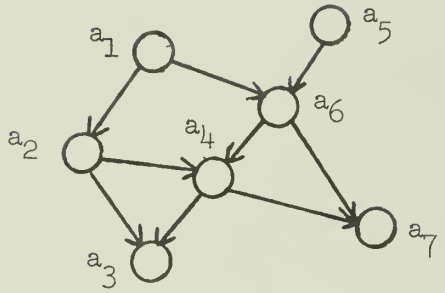


Figure 4. An Example of Ascending Level Assignment

$$A(G) = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Using above mentioned process, A(G) can be rearranged as the following:

$$A(G) = \begin{array}{c} \\ a_1 \\ a_5 \\ a_2 \\ a_6 \\ a_4 \\ a_3 \\ a_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} a_1 & a_5 & a_2 & a_6 & a_4 & a_3 & a_7 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

From this matrix,

$$A(0) = \{ a_1, a_5 \}$$

$$A(1) = \{ a_2, a_6 \}$$

$$A(2) = \{ a_4 \}$$

$$A(3) = \{ a_3, a_7 \}$$

4.2.2 Determination of All Nodes Bounded by Maximal Strongly Connected Subgraphs .

From the definition of ascending and descending level assignment, the set L of nodes which are bounded by maximal strongly connected subgraph is obtained by

$$L \subset X - \left(\bigcup_{i=o}^{n} A(i) \cup \bigcup_{i=o}^{m} D(i) \right)$$

where n and m are the largest level in ascending and descending level respectively.

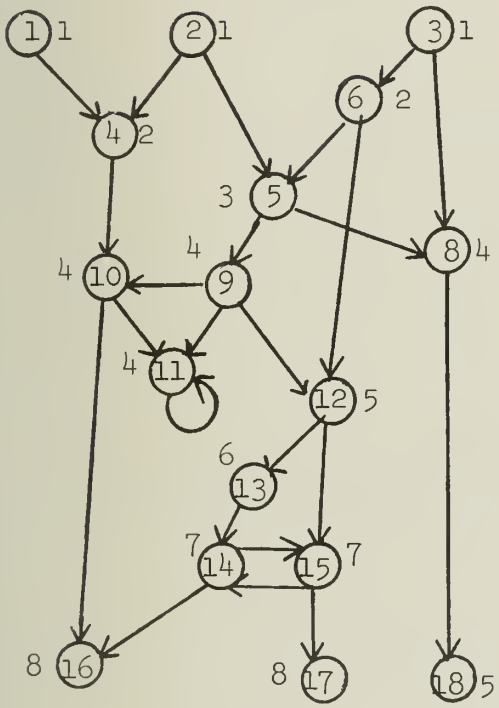
4.2.3 Level Assignment for Any Graph

In general, a directed graph may have some strongly connected subgraphs and some nodes may be bounded by strongly connected subgraphs. In this case, as stated in Section 4.2.2, the levels cannot be assigned to these nodes. However, the idea of the level assignment can be generalized from link graph to all directed graph.

Theorem 2. Every node in any directed graph can be assigned a level if the nodes of a maximal strongly connected subgraph are all assigned the same level.

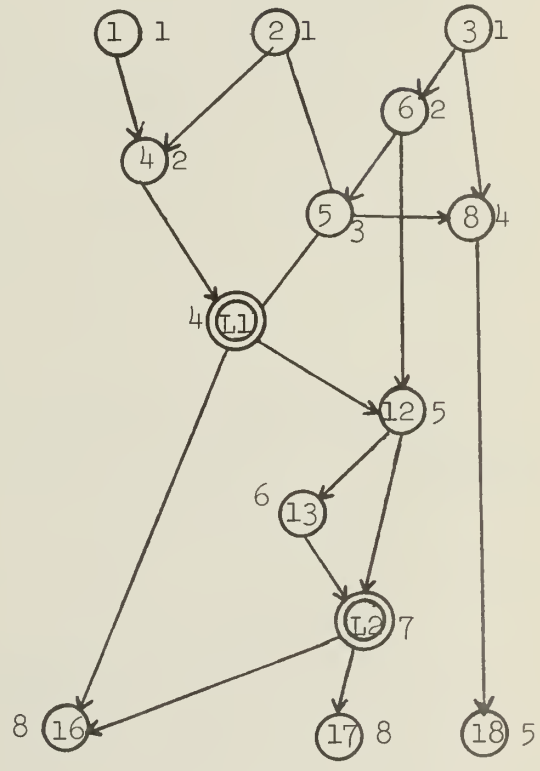
Proof: If the nodes of maximal strongly connected subgraph are assigned the same level, these nodes can be treated as one node. When this is done, the resulting condensed graph has no loops; therefore all nodes can be assigned to a level by the definition of the level assignment.

Example 2. Level assignment of any Directed Graph.



(a)

Example of a directed graph having maximal strongly connected subgraph.



(b)

The condensed graph of the example.

Figure 5. Level Assignment of Any Directed Graph

In Figure 5a, nodes 9, 10, 11 and nodes 14, 15 constitute maximal strongly connected subgraphs and nodes 12 and 13 are bounded by these maximal strongly connected subgraphs. In the corresponding condensed graph, the first maximal strongly connected subgraph is replaced by the node L_1 and the second by L_2. Those represented by double circles is shown in Figure 5b, in which the integer next to a node represents level. Figure 5a shows the complete ascending level assignment derived from Figure 5b.

4.3 Loop Detection

In order to assign the levels for any directed graph, it is necessary to condense the graph to a link graph. The condensation can be achieved after maximal strongly connected subgraphs are detected. Determination of these subgraph can be made from using the reachability matrix.

4.3.1 Determination of R matrix

Theorem 3. The Reachability matrix is described by the following recursion relationships:

$$R_1(G) = A(G)$$
$$R_n(G) = (A(G) \times R_{n-1}(G)) \cup R_{n-1}(G)$$
$$R(G) = R_p(G) \qquad n = 2, 3, \ldots p.$$

where p is the number of nodes in a graph G.

Multiplication and inclusive or in Theorem 3 is defined by the following formula, where X and Y are matrices of order n.

$$X \times Y = [x_{ij}] \times [y_{ij}] = [\bigcup_{k=1}^{n} x_{ik} \cdot y_{kj}]$$
$$X \cup Y = [x_{ij}] \cup [y_{ij}] = [x_{ij} \cup y_{ij}]$$

Proof: The proof of Theorem 3 is divided into two parts. The first is to show that the equation represents a process for obtaining the reachability matrix. The second is that the number of repetitions, p, is a sufficient condition for obtaining the reachability matrix.

Part 1. The recursion formula can also be represented by

$$R_p(G) = A \cup A^2 \cup A^3 \cup \ldots \cup A^p$$

Since $A^2 = [a_{ij}^{(2)}] = [\bigcup_{k=1}^{p} a_{ik} \cdot a_{kj}]$, element $a_{ij}^{(2)} = 1$ if there is at least one path of length 2 from a_i to a_j i.e.,

$$a_{ij}^{(2)} = \begin{cases} 1 & : \text{ if } a_j \in \Gamma^2 a_i \\ 0 & : \text{ if } a_j \in \Gamma^2 a_i \end{cases}$$

Therefore, A^n represents a matrix whose elements $a_{ij}^{(n)}$ indicate whether or not there is at least one path of length n from a_i to a_j depending upon whether $a_{ij}^{(n)} = 1$ or $a_{ij}^{(n)} = 0$, respectively. Hence, the element of γ_{ij} of $R(G) = \bigcup_{i=1}^{p} A_i$ is 1 <u>if and only if there exists at least one path of length equal to or less than p from a_i to a_j.</u>

Part 2. Whenever a_j is reachable from a_i in a given graph with p nodes, there must be a path of length at most p from a_i to a_j.

In other words, since there are only p nodes available, any path cannot be of length greater than p, thus proving the theorem.

From the Theorem 3, the following corollary is obtained:

Corollary 1. Let k be the minimum value of n such that $R_n(G) = R_{n-1}(G)$ for $n \leq p$ with p nodes graph. Then $R_k(G) = R(G)$.

Proof: If the length of all possible paths is less than or equal to k in graph G, $R_k(G)$ includes all possible paths.

Hence $A(G) \times R_k(G) = R_k(G)$

$$R_{k+1}(G) = R_k \cup (A(G) \times R_k(G)) = R_k(G) \cup R_k(G) = R_k(G) = R(G)$$

4.3.2 Determination of Maximal Strongly Connected Subgraphs

By definition, a maximal strongly connected subgraph is a subgraph with the maximal nodes which are mutually reachable. Hence the following theorems follow:

Theorem 4. If an element s_{ij} of the elementwise product $R(G) \cdot R^T(G)$ is 1, nodes a_i and a_j are members of a strongly connected subgraph.

Proof: The elementwise product of $R(G) \cdot R^T(G)$ is defined by:

$$S(G) = R(G) \cdot R^T(G) = [\gamma_{ij}] \cdot [\gamma_{ji}] = [\gamma_{ij} \cdot \gamma_{ji}]$$

This equation means that an element s_{ij} of S(G) is 1, if a_j is reachable from a_i and a_i is reachable from a_j.

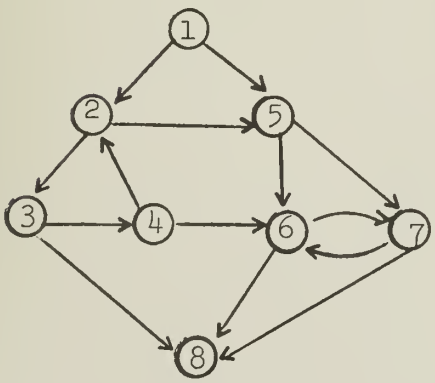
Theorem 5. S(G) is symmetric.

This is clear from the definition of S(G).

Theorem 6. In the S matrix, if a diagonal element is non-zero, then the corresponding node is a member of a loop. The maximal loop, which is equivalent to the set of nodes of a maximal strongly connected subgraph, can be obtained from all distinct non-zero elements in a row.

Proof: If a diagonal element s_{ii} of the S matrix is 1, then the corresponding node is a member of a loop since there exists a path from node a_i to a_i. The members of a maximal strongly connected subgraph are those nodes whose corresponding elements in S(G) are 1. If $S_{ii} = 1$, non-zero elements in the i^{th} row must have a path both to and from node a_i, while no such path-pair exists for zero elements. Hence, all distinct non-zero elements in a row constitute a maximal strongly connected graph.

Example 3. Determination of a maximal strongly connected graph.



$$A(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left[\begin{array}{cccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Figure 6. An Example of the Determination of Maximal Strongly Connected Graphs in G

$$A^2(G) = A(G) \times A(G) =$$

	1	2	3	4	5	6	7	8
1	0	0	1	0	1	1	1	0
2	0	0	0	1	0	1	1	0
3	0	1	0	0	0	1	0	0
4	0	0	1	0	1	0	1	1
5	0	0	0	0	0	1	1	1
6	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0

$$R_2(G) = A(G) \cup A^2(G) =$$

	1	2	3	4	5	6	7	8
1	0	1	1	0	1	1	1	0
2	0	0	1	1	1	1	1	0
3	0	1	0	1	0	1	0	1
4	0	1	1	0	1	1	1	1
5	0	0	0	0	0	1	1	1
6	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0

$$A^3(G) = A(G) \times A(G) \times A(G) =$$

	1	2	3	4	5	6	7	8
1	0	0	0	1	0	1	1	1
2	0	1	0	0	0	1	1	1
3	0	0	1	0	1	0	1	1
4	0	0	0	1	0	1	1	1
5	0	0	0	0	0	1	1	1
6	0	0	0	0	0	0	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0

$R_3(G) = A^3(G) \cup R_2(G) = R_4(G) = R(G) =$

	1	2	3	4	5	6	7	8
1	0	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1
5	0	0	0	0	0	1	1	1
6	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0

$$S(G)=R(G)\cdot R^{T}(G)=\begin{bmatrix} 0&1&1&1&1&1&1&1\\ 0&1&1&1&1&1&1&1\\ 0&1&1&1&1&1&1&1\\ 0&1&1&1&1&1&1&1\\ 0&0&0&0&0&1&1&1\\ 0&0&0&0&0&1&1&1\\ 0&0&0&0&0&1&1&1\\ 0&0&0&0&0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&0 \end{bmatrix}$$

$$= \begin{array}{c} \\ 1\\ 22\\ 33\\ 44\\ 55\\ 66\\ 77\\ 88 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1&2&3&4&5&6&7&8 \end{array}\\ \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&1&1&1&0&0&0&0\\ 0&1&1&1&0&0&0&0\\ 0&1&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix} \end{array}$$

From S(G). the graph G has two maximal strongly connected subgraphs: the one consists of the nodes 2, 3 and 4, and the other consists of the nodes 6 and 7.

4.4 Ordering

When a loop is detected in a given graph, the logical equations of the corresponding nodes (IC packages) are put in a program loop and executed until the values of those nodes in the loop are all stabilized. The efficiency of stabilizing the program loop of the simulation depends on the order of nodes in a loop. If the graph is re-drawn so that its nodes, with the associated branches, are ordered in sequence from top to bottom and if a branch whose arrow points from top to bottom is called a normal branch and a branch whose arrow points from bottom to top is called a reverse branch, then the following property can be heuristically stated.

If nodes are ordered so that the number of reverse branches is minimized, the number of iterations necessary to stabilize a program loop will become minimized provided that the node in the top position has inputs from outside.

Since the evaluation of the nodes in a loop is made from top to bottom in a program, only the nodes which have incoming normal branch can be defined in the first iteration. Then in the following iteration, at least one node having an incoming reverse branch will be defined since the nodes which have incoming branches from defined nodes can produce the correct outputs. Therefore, if the order of the nodes is such as to produce the minimum number of reverse branches, the number of iteration will be minimized.

The directed graph representation is not sufficient to calculate the necessary number of iterations, but logical connections within a graph must also be given.

For example, if a loop consists of four nodes as shown in Figure 7a and its order is defined such as shown in Figure 7b, there are four reverse

branches in the order. If the logical connection of the graph is as shown in Figure 7c, four iterations are necessary to stabliize the loop. That is, in the first iteration, the values associated with nodes 1a and 2a are defined, in the second, the value of 3a is defined, in the third, the value of 4 and finally 1b, 3b and 2b will be defined.

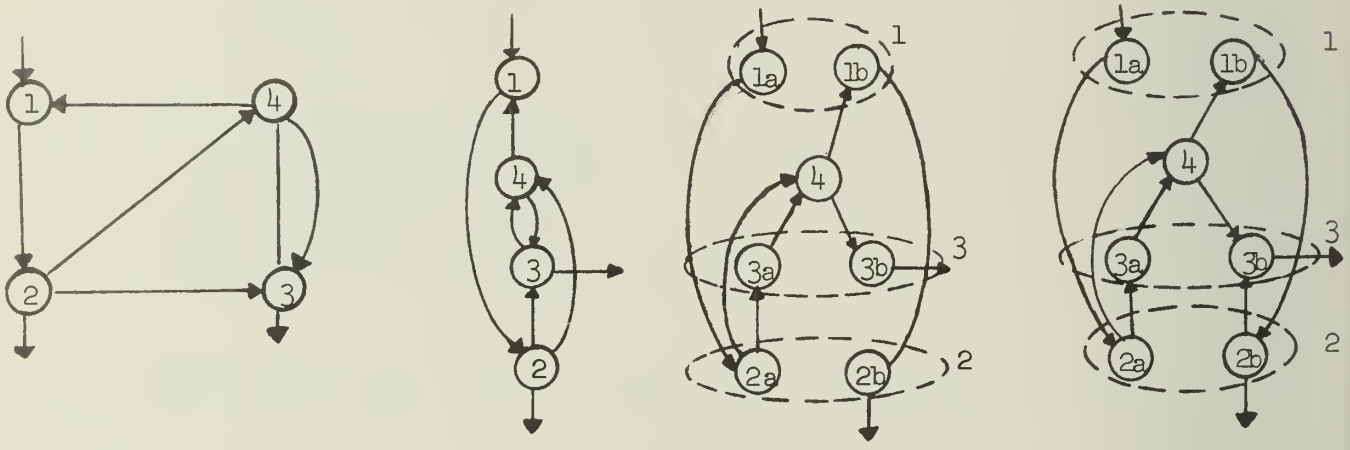


Figure 7. An Example of Node Ordering and Its Logical Connections

However, if the logical connection is given as shown in Figure 7d, five iterations are necessary, i.e., 3b is stabilized in the fifth iteration. Therefore, it is not possible to determine the number of necessary iterations even though the order of nodes and its branches are fixed.

Finding the ordering having the minimum number of reverse branches is difficult in the general case unless all permutations are examined. However, optimal solution for this problem can be heuristically stated as follows:

The most efficient order of nodes is along the longest path within the loop which does not pass through any node twice.

Consider the previous example, in which the most efficient order is that in which the nodes are ordered as shown in Figure 8a, which is along the longest path. The logical connections of the example are shown in Figures 8b, and 8c. In these cases only two iterations are necessary.

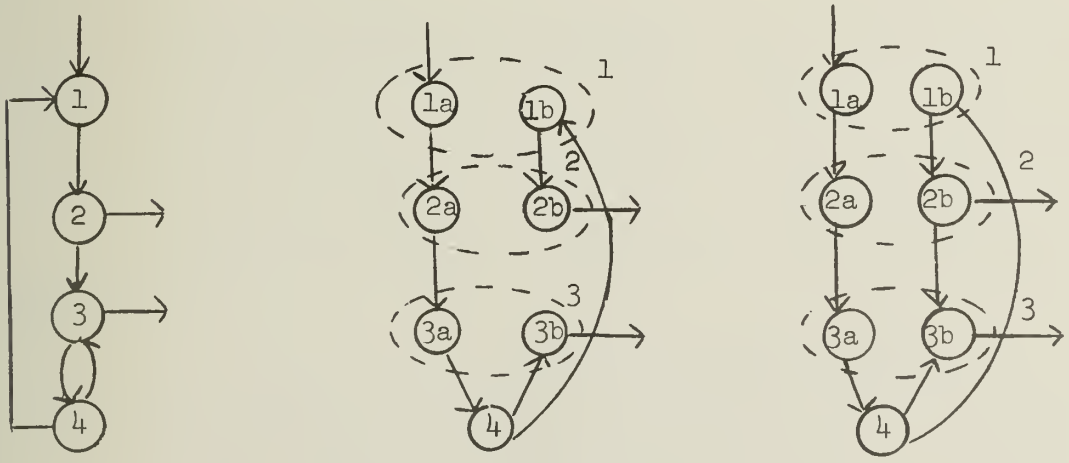


Figure 8. The Modified Order and Its Logical Connections.

4.4.1. Determination of the Longest Paths in a Graph [6].

A method of finding the longest path or paths in a graph can be obtained by modifying a reachability matrix such that the matrix contains path information. The following definitions are necessary:

Definition 1.

A path matrix* of length n is defined as follows:

$$P^{(n)}(G) = [p^{(n)}_{ij}]$$

$$p^{(n)}_{ij} = \begin{cases} \text{sequence(s) of n+1 distinct node names starting from node i and ending at node j if there exists such path(s) of length n} \\ 0 \quad \text{if no such path exists} \end{cases}$$

From this matrix, another matrix $p^{(o)}(G)$ called initial path matrix is deduced by removing the first node name in each element from the matrix $P^{(1)}(G)$ of length 1.

Definition 2.

Let A be $P^{(n)}(G)$ and let B be $P^{(o)}(G)$.

Then product AB of multiplication over these path matrices is defined as follows:

The operation is the same as the usual method of "line and column." Whenever $p^{(n)}_{ij}$ of $P^{(n)}(G)$ is multiplied by $p^{(o)}_{ij}$ of $P^{(o)}(G)$, zero

---

* This matrix is called a Latin matrix in graph theory.

results in the (i,k) position of the resulting path matrix AB if either or both of these element contains a zero. A zero is also entered in the position if from all the products, it is not possible to obtain a sequence in which no nodes are repeated when a sequence of $p^{(o)}_{jk}$ of $P^{(o)}(G)$ is placed after a sequence of $p^{(n)}_{ij}$ of $P^{(n)}(G)$. If a sequence of $p^{(o)}_{jk}$ can be placed after a sequence of $p^{(n)}_{ij}$ and one or more sequences are found in which no node is repeated, the sequence (or sequences) is put into the (i,k) position of the resulting matrix.

By this definition, the resulting matrix is obviously a path matrix of length n+1. Hence, the path matrix of length n can be obtained by the recursion formula:

$$P^{(n)}(G) = P^{(n-1)}(G) \dot{\times} P^{(o)}(G)$$

where $\dot{\times}$ denotes the above mentioned multiplication.

The method of finding the longest path(s) in a graph is then described by

$$P^{(n+1)}(G) = P^{(n)}(G) \dot{\times} P^{(o)}(G) \qquad n = 1,2,3. \ldots$$

If $P^{(n+1)}(G) = 0$ or $n+1 = m$, where m is the number of nodes of a given graph, then $P^{(n)}(G)$ contains the longest paths.

Example 4. The determination of the longest path(s) of Figure 9.

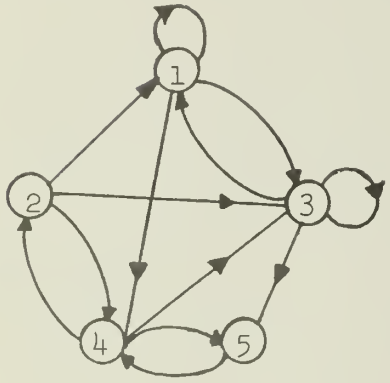


Figure 9. An Example of the Determination of the Longest Paths.

$$P^{(0)}(G) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 3 & 4 & 0 \\ 2 & 1 & 0 & 3 & 4 & 0 \\ 3 & 1 & 0 & 0 & 0 & 5 \\ 4 & 0 & 2 & 3 & 0 & 5 \\ 5 & 0 & 0 & 0 & 4 & 0 \end{array}$$

$$P^{(1)}(G) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 13 & 14 & 0 \\ 2 & 21 & 0 & 23 & 24 & 0 \\ 3 & 31 & 0 & 0 & 0 & 35 \\ 4 & 0 & 42 & 43 & 0 & 45 \\ 5 & 0 & 0 & 0 & 54 & 0 \end{array}$$

$$P^{(2)}(G) = P^{(1)}(G) \dot{\times} P^{(0)}(G) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 142 & 143 & 0 & 135 \\ & & & & & 145 \\ 2 & 231 & 0 & 213 & 214 & 235 \\ & & & 243 & 314 & 245 \\ 3 & 0 & 0 & 0 & 354 & 0 \\ 4 & 421 & 0 & 423 & 0 & 435 \\ & 431 & & & & \\ 5 & 0 & 542 & 543 & 0 & 0 \end{array}$$

$$P^{(3)}(G) = P^{(2)}(G) \dot{\times} P^{(0)}(G) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 1423 & 1354 & 1435 \\ 2 & 2431 & 0 & 2143 & 2314 & 2135 \\ & & & & 2354 & 2435 \\ & & & & & 2145 \\ 3 & 0 & 3142 & 0 & 0 & 3145 \\ & & 3542 & & & \\ 4 & 4231 & 0 & 4213 & 0 & 4235 \\ 5 & 5421 & 0 & 5423 & 0 & 0 \\ & 5431 & & & & \end{array}$$

$P^{(4)}(G) = P^{(3)}(G) \times P^{(o)}(G) =$

	1	2	3	4	5
1	0	13542	0	0	14235
2	0	0	0	21354	21435 23145
3	35421	0	0	0	0
4	0	0	0	0	42135
5	54231	0	54213	0	0

In this example, nine longest paths can be obtained; they are

13542, 14235, 21354, 21435, 23145, 35421,
42135, 54231, 54213.

Although this method is quite clear, it is not suitable for a computer. From this idea, however, a suitable algorithm for a computer can be obtained.

4.4.2 An Algorithm to Find the Longest Paths.

An algorithm suitable for a computer can be stated as follows, assuming that each node is represented by a distinct integer:

Step 1. Establish $P^{(o)}(G)$ matrix.

Step 2. Construct a table of paths of length 1 so that each path row is in numerical order.

Step 3. Execute the following procedure for each row of the table. Examine each element from the row of $P^{(o)}(G)$ which corresponds to the rightmost integer of a path. For all non-zero elements in the $P^{(o)}(G)$

matrix, compare the value of the element with each integer of the path, and if the value does not agree with any of the elements of the path, put it into the next column of the path. If another path can be found, place the entire new path in the empty row of the table. If, however, the matrix value already appears in the path, then cross out that path.

Step 4. Repeat Step 3 until all rows are crossed out or the number of iterations becomes equal to the number of nodes in a given graph minus one.

Step 5. If the termination is made by all rows crossed out, extract those paths which were crossed out at the last iteration. If the termination is made without all rows crossed out, extract paths which were not crossed out. These paths are the longest ones.

Using this algorithm, the previous example can be solved as follows:

Example 5.

$P^{(0)}(G) =$

	1	2	3	4	5
1	0	0	3	4	0
2	1	0	3	4	0
3	1	0	0	0	5
4	0	2	3	0	5
5	0	0	0	4	0

PATH TABLE

Paths of Length 1	2	3	4
21	3	5	4
31	4	2	×
42	1	3	5
13	5	4	2
23	1	4	5
43	1	×	
14	2	3	5
24	3	1	×
54	2	1	3
35	4	2	1
45	×		
21	4	3	5
42	3	1	×
23	5	4	×
43	5	×	
14	3	5	X
14	5	×	
24	5	×	
54	3	1	×
31	4	5	×
24	3	5	×
54	2	3	1
21	4	5	×
42	3	5	×

From the table, the longest paths are:

21354, 42135, 13542, 23145, 14235, 54213, 35421, 21435, and 54231. Those are the same as in Example 4.

4.5 Algorithm for Generating the Simulator Body

The algorithm for generating the simulator body, which consists primarily of level assignment, loop detection and longest paths determination is defined by the following steps. Before starting from Step 1, it is assumed that the wirelist has been converted to a graph description, i.e., a branch list is provided.

Step 1. Assign levels to packages in an ascending order as far as possible.

Step 2. Assign levels to packages in a descending order as far as possible.

Step 3. Extract the unassigned packages (these are bounded by loops).

Step 4. Construct the adjacency matrix for the packages extracted above.

Step 5. Get the reachability matrix.

Step 6. Get S matrix and determine all maximal strongly connected subgraphs.

Step 7. Get the longest path in each maximal strongly connected subgraph and get a sequence of packages so that the first package of the sequence has inputs fron outside of the loop.

Step 8. Get the condensed graph, in which the packages of maximal strongly connected subgraphs are represented by one pseudo-package so that the modified graph has no loops.

Step 9. Assign a level to each package in the condensed graph in ascending order.

Step 10. Expand the pseudo-packages to the original ones using the sequence obtained in Step 7 and reassign a level to all packages in an ascending order such that

i) The level of packages which were assigned by Step 1 are less than the levels of looped packages.

ii) The levels of loop packages are less than the level of the packages which were assigned in Step 2.

Step 11 Using the sequence obtained above, generate a set of logical equations for each package by the following rules:

i) Referring to the levels assigned to the packages and to the connection list, generate sets of logic equations whose independent variables are elements corresponding to signals in the connection list.

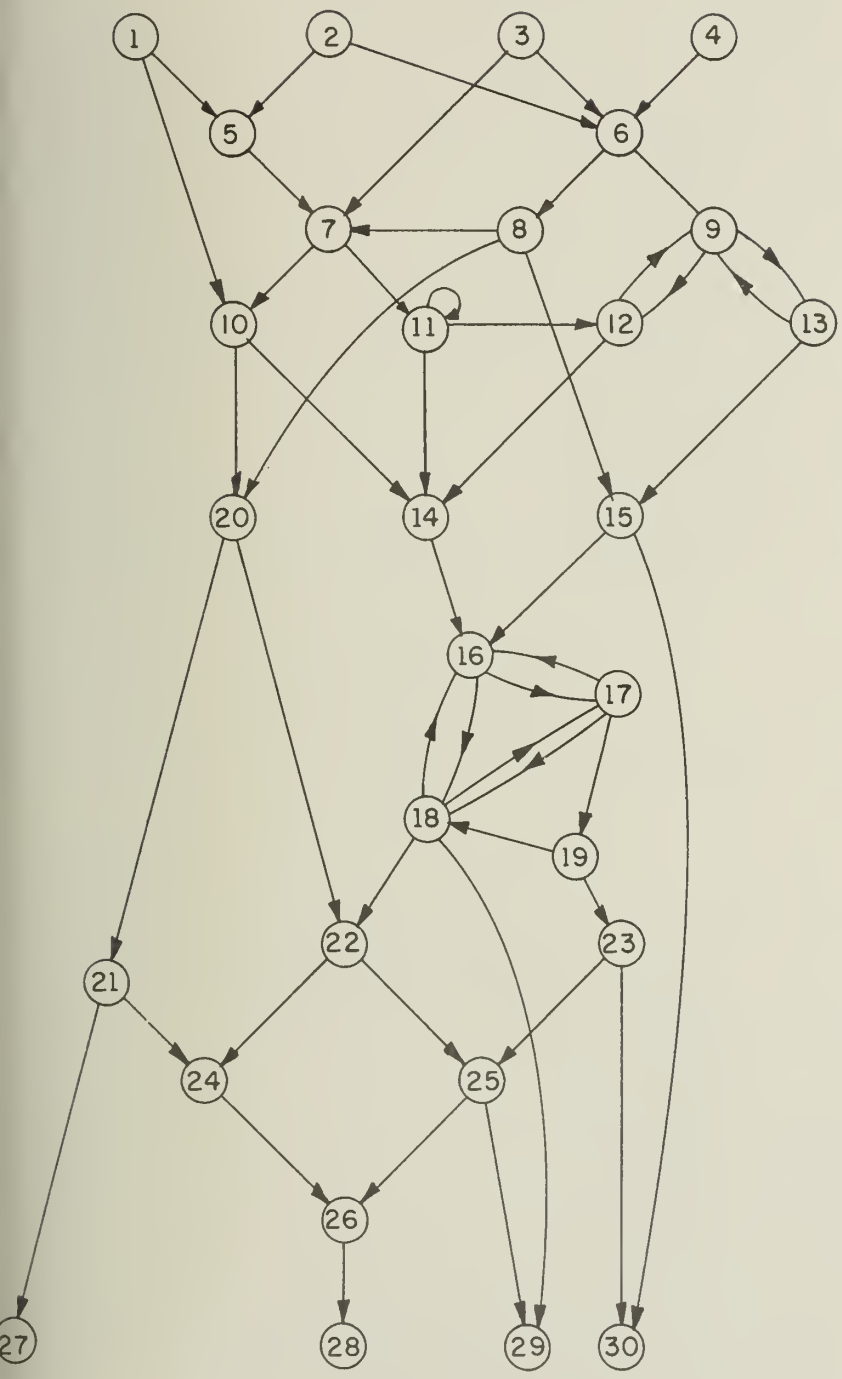
ii) For a storage package of the first level, generate only equations which correspond to the output part.

iii) For a storage package of the last level, generate only equations which correspond to the input part.

iv) For each loop, add program loop statements and statements which test for stability by recognizing stabilized output values from the equations.

One example of these steps is shown in Example 6.

Example 6.



Step 1. Ascending level assignment.

level	nodes
0	1, 2, 3, 4
1	5, 6
2	8
3	7
4	10
5	20
6	21
7	27

Step 2. Descending level assignment

level	nodes
0	27, 28, 29, 30
1	26
2	24, 25
3	21, 22, 23
4	20

Figure 10. An Example of a Graph for the Simulator Body Generator.

Step 3. Extracted nodes are shown in Figure 11.

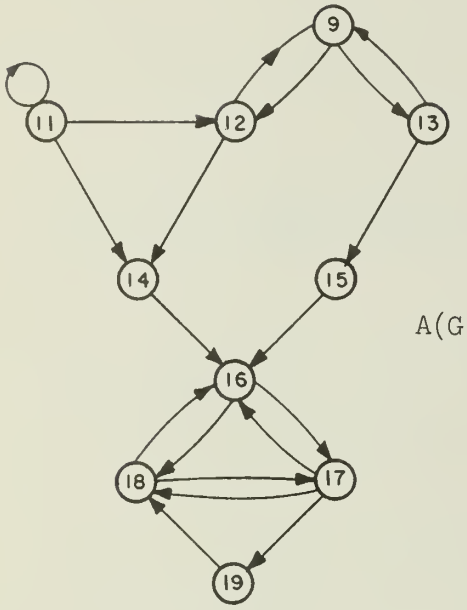


Figure 11. Extracted Graph Bounded by Loops

Step 4. Adjacency Matrix A(G)

$$A(G) =$$

	9	11	12	13	14	15	16	17	18	19
9	0	0	1	1	0	0	0	0	0	0
11	0	1	1	0	1	0	0	0	0	0
12	1	0	0	0	1	0	0	0	0	0
13	1	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	1	1	0
17	0	0	0	0	0	0	1	0	1	1
18	0	0	0	0	0	0	1	1	0	0
19	0	0	0	0	0	0	1	0	0	0

Step 5. Reachability Matrix R(G)

$$R(G) = R_5(G) = R_6(G) =$$

	9	11	12	13	14	15	16	17	18	19
9	1	0	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1
12	1	0	1	1	1	1	1	1	1	1
13	1	0	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	1	1	1	1
15	0	0	0	0	0	0	1	1	1	1
16	0	0	0	0	0	0	1	1	1	1
17	0	0	0	0	0	0	1	1	1	1
18	0	0	0	0	0	0	1	1	1	1
19	0	0	0	0	0	0	1	1	1	1

Step 6.

$$S(G) = R(G) \times R^{T}(G) =$$

	9	11	12	13	14	15	16	17	18	19
9	1	0	1	1	0	0	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0
12	1	0	1	1	0	0	0	0	0	0
13	1	0	1	1	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	1	1	1	1
17	0	0	0	0	0	0	1	1	1	1
18	0	0	0	0	0	0	1	1	1	1
19	0	0	0	0	0	0	1	1	1	1

By S(G), the graph has three loops:

loop 1 = {9, 12, 13}
loop 2 = {11}
loop 3 = {16, 17, 18, 19}

Step 7.

a) The longest path for loop 1.

$$P^{(o)}(G) =$$

	9	12	13
9	0	12	13
12	9	0	0
13	9	0	0

1	2
9,12	×
9,13	×
12,9	13
13,9	12

The longest paths are 12, 9, 13 and 13, 9, 12.

b) The longest path for loop 2.

Since the loop is formed from only one node, there is no path.

c) The longest path for loop 3.

$$P^{(o)}(G) = \begin{array}{c} \\ 16 \\ 17 \\ 18 \\ 19 \end{array} \begin{array}{c} \begin{array}{cccc} 16 & 17 & 18 & 19 \end{array} \\ \begin{bmatrix} 0 & 17 & 18 & 0 \\ 16 & 0 & 18 & 19 \\ 16 & 17 & 0 & 0 \\ 16 & 0 & 18 & 0 \end{bmatrix} \end{array}$$

1	2	3
16,17	18	×
16,18	17	19
17,16	18	×
17,18	16	×
17,19	16	18
18,16	17	19
18,17	16	×
19,16	17	18
19,18	16	17
16,17	19	18
17,19	18	16
18,17	19	16
19,16	18	17
19,18	17	16

The longest paths are:

16, 18, 17, 19; 17, 19, 16, 18
18, 16, 17, 19; 19, 16, 17, 18
19, 18, 16, 17; 16, 17, 19, 18
17, 19, 18, 16; 18, 17, 19, 16
19, 16, 18, 17; 19, 18, 17, 16

The sequence of the nodes for loop 1 is thus 12, 9, 13.

The sequence of the nodes for loop 3 is 16, 18, 17, 19 (or 16, 17, 19, 18).

Step 8. The condensed graph is shown in Figure 12.

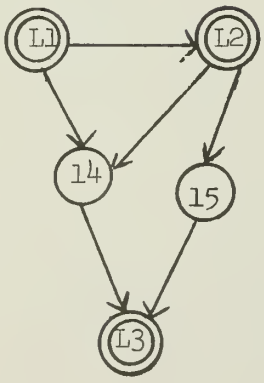


Figure 12. Condensed Graph of the Example.

Step 9. Ascending level assignment

Level 0 = {L1}

Level 1 = {L2}

Level 2 = {14,15}

Level 3 = {L3}

Step 10. The total level assignment and the order of nodes within a level can be obtained as follows:

We assume that the nodes assigned by the ascending ordering are leveled starting from level 0. The nodes bounded by loops start from level 50, and the nodes from the last level are assigned to level 100.

level 0	1, 2, 3, 4
1	5, 6
2	8
3	7
4	10
5	20
6	21
7	27
50	11
51	12, 9, 13
52	14, 15
53	16, 18, 17, 19
96	20
97	21, 22, 23
98	24, 25
99	26
100	27, 28, 29, 30

Step 11. The logical equations will be generated in the above order. Program loops and test statements will be inserted in levels 50, 51 and 53. For the actual logic equations, see Appendix G1.

5. SIMULATION CONTROL LANGUAGE (TESLA)

TESLA - TEst Simulator LAnguage - is a fairly general logic simulation input, output and control language. Language statements consist of two categories: declarations, and test step commands. Statements are written in free-field format separated by semicolons. This section presents a short description of the language and a sample program to provide an introduction of the language. The complete syntax and semantics of this language are described in [2].

5.1 Declaration

5.1.1 Program

A TESLA program starts with the word BEGIN. The first declaration must be a case limit declaration, which specifies the range of test cases. The other declarations may be freely written in a program, provided that all quantities are declared before they are used. Redeclarations are permissible.

A step label followed by a series of statements assigns values to signals and controls the simulation in one clock interval.

A program must end with the word END.

5.1.2 Declarations

a) Case limit declaration

CASES [n:m] - this specifies the number of test cases to be executed in parallel. m and n are integers and $0 \leq m - n \leq 46$.

b) Group declarations

SIGNAL GROUP - used to associate an identifier with a group of signals. Any signal name may be used in the list.

STORAGE GROUP - used to associate an identifier with a group of storage signals. In this case, the next state of storage element is assigned to the signal.

A multi-signal identifier, which is written by using *'s as "don't cares" in place of characters, specifies a group of signals whose names match the non-asterisked portions of the multi-signal identifier.

Group declarations may be nested to any depth.

c) Data declarations

DATA - associates bit patterns with an identifier. Bit patterns may be expressed either in octal or binary form. If bit patterns are case dependent, case dependent digits (A ~ Z) may be used in the bit pattern, provided they have been declared.

d) Digit Declaration

DIGIT - associates digit patterns with an identifier which must be single letter of the alphabet. The digit is called a case dependent digit. This declaration is used to declare digits whose value varies with the test case. The first digit of the digit list is associated with the lower limit of the simulation cases declaration, the next digit is associated with the next case, etc. This case dependent digit may be declared in octal or binary form.

5.2 Control Statement

There are three categories: input control statements, storage control statements and output control statements. A step label followed by any control statement and/or declarations specifies values of signals and control of the simulator and its output.

5.2.1 Input Control Statements

Three simulation input control commands may be used. These commands may adjust input variables of specified test cases among those being generated in parallel by specifying case limit.

INSET - causes each of the input variables associated with the list of input signals following the command to be set to a logical "one".

INCLR - causes each of the input variables associated with the list of input signals following the command to be set to a logical "zero".

INPUT - causes the input variables associated with the list of input signals to be modified by a specified data pattern.

INOUT - permits input signal values to be set to the values of the values of signals calculated during the previous test step.

The identifier ALL and the (default) empty word both cause the specified action to be applied to all input variables.

5.2.2 Storage Element Control Statements

Three commands can be used to control simulated storage elements. Case limits can be used to restrict storage element control statements to apply to speicified test cases.

FFX - causes the storage elements to be set their next states in accordance with values calculated during the previous test step. This corresponds to the actual operation of the physical storage element. Therefore, unless otherwise specified by an explicit storage element control statement, the transfer (FFX) condition is assumed to apply to all storage elements.

FFIN - similar to the INPUT command. It causes sets of storage elements to assume the states given by an associated bit pattern. This bit pattern is used instead of the normal next storage element states.

FFOLD - this inhibits the transfer of calculated next storage element states into the present states and causes the storage element states to remain unchanged from those held during the previous test step.

It is assumed that at the start of simulation, all storage elements will be initially cleared to logical zero.

5.2.3 Output Control Statements

These statements are used to print the simulation results with various formats. These commands may include case limits to selectively printout the results of only some of the many test case results which have been generated in parallel.

PRINT2 or PRINT8 - causes printing of the simulation output in binary or octal form for the list of output signals and/or storage elements following the command.

PRAC2 or PRAC8 (PRint And Compare) - causes the printing (in binary of octal) of not only the simulation output for each member of the list following the command, but also an expected output pattern which has been associated with each of these lists. If differences exist between this expected output and the actual output, a marker is printed which directs the observer's attention to the location of the differences.

PRID2 or PRID8 (PRint If Different) - has the same statement format as PRAC and causes the same type of output but only if there are differences between the actual and expected outputs. If the simulation output is as expected, no printout occurs.

5.3 A Sample Program

An example of a TESLA program is given in Figure 13 which simulates the logic of one of Control Unit boards. The following descriptions will hopefully clear up some of the confusion of the previous sections.

line

2 The first declaration must be the number of test cases which are generated in parallel.

3-6 The group identifier TOM is associated with sixteen input connector signal names which are written within a pair of parenthesis.

7-8 These group declarations associate multi-signal identifiers with TGM, ABT, AGT, and AGO respectively. The group identifier TOM could also have been declared using a multi-signal identifier as follows:

```
SIGNAL    GROUP    TOM(TOM-**--G1);
```

9 This declaration is for storage element group FF.

10-13 These are the same as in line 3-8 except they associate output connector signal names with the identifiers which are used for print statements.

14-17 These digit declarations associate the given letters with the case dependent digit patterns. The letter D, for example, is associated with a case dependent digit whose value is zero in the odd numbered test cases and one in the even numbered ones.

18 PAT 1 is a sixteen-bit pattern. The first four pins are set to different values for the different parallel test cases according to the digit declaration A. The second four pins are set according to the digit declaration B, the third four to C and the last four to D. For example, the first pin is set to zero for test cases 1 through 8 and one for test cases 9 through 16.

19-20 PAT2 and PAT3 are eighteen bit patterns and declared using octal notation.

21-22 PAT4 and PAT5 are declared using binary notation.

23 The first test step of this simulation is called STEP1. The values of the group of storage elements identified with FF are all set to zero for test cases 1 through 4, all ones for test cases 5 through 8 and first eight signal values are set to ones and second eight signal values are set to zeroes for test cases 9 through 12 according to PAT3 and so on.

24 Since there are no case limits, for all cases the signals CLK-----C0 is set to one and CLK-----C1 is set to zero.

25 The first statement prints the values of the members of FF in binary form and the second statement prints the values of all output connector signals.

28 The members of TOM and TGM are set according to the pattern PAT1.

29 The values of the members of TR0, COMP, TGC and FF are printed in binary form.

35 "NOT PAT1" means the Boolean complement of the values of PAT1. Therefore, the members of AGT are set to the inverses of the values of PAT1.

36 The third statement is a print-if-different in binary form. If the expected values of FF for all cases are one, no printout occurs. However, if there is difference, the expected and actual values and the points of difference are printed out.

37 The first statement prints all output connector signal values. This is equivalent to PRINT2 ALL. The second is a print-and-compare in binary form. The values of the members of COMP for test case sixteen are compared with ones and the expected (in this case all ones) and actual values and the points of difference (which are indicated by X's under the differences) are printed out.

41 By this statement STEP10 is repeated twice.

47 END. terminates the program.

```
                    ILLIAC IV TRANSLATOR WRITING SYSTEM
          TESLA   COMPILER - VERSION  1    DECEMBER 22, 1969  AT   0: 2:48.9
                    PROGRAM QTOFUND/TESLA


$RSWD TIMES
BEGIN                                                                    1
CASES[1:16];                                                             2
     SIGNAL GROUP TIM(TDM=12--G1,TDM=13--G1,TDM=14--G1,TDM=15--G1,       3
                      TDM=28--G1,TDM=29--G1,TDM=30--G1,TDM=31--G1,       4
                      TDM=44--G1,TDM=45--G1,TDM=46--G1,TDM=47--G1,       5
                      TDM=60--G1,TDM=61--G1,TDM=62--G1,TDM=63--G1),      6
                 TCL(TCL=****--G0), ART(ART=****--G0),                   7
                 AGT(AGT=****--G0), AGD(AGD=****--G0);                   8
     STORAGE GROUP FF(TRD=****--[1]);                                    9
     SIGNAL GROUP TRC(TRD=****--[0),                                    10
                 COMP(T=COMPOK,TCOMPCHK=7,TCOMPCHK=8,T=COMPD=G1),       11
                  TCC(TGC=28--G1,TGC=29--G1,TGC=30--G1,TGC=31--G1,      12
                      TGC=44--G1,TGC=45--G1,TGC=46--G1,TGC=47--G1);     13
     BIGIT A=2(000000001111111),                                        14
           B=2(000011110000111),                                        15
           C=2(001100110011001),                                        16
           D=2(010101010101011);                                        17
     DATA PAT1=2(AAAAFFFF0000FFFF),                                     18
          PAT2=B(000000),                                               19
          PAT3=B(777777),                                               20
          PAT4=2(1111111100000000),                                     21
          PAT5=2(0000000011111111);                                     22
STEP1:    FFIN FF [1:4] PAT2,[5:8] PAT3,[9:12] PAT4,[13:16] PAT5;       23
          INSET CLK-----C0; INCLR CLK-----C1;                           24
          PRINT2 FF: PRINT2 ALL;                                        25
```

Figure 13. An Example of TESLA Program

```
STEP2:    INSET T-TRDE-KG1,T-TRDG-KG1,T-CLIN--B1,CLK-----C1,T-TRD4-KG1,  26
               T-COMPC-G1;INCLR CLK-----C0;                               27
          INPUT TGM PAT1,TGM PAT1;                                        28
          PRINT2 TRC,COMP,TGC,FF;                                         29
STEP4:    INSET TGTTRDEN-1;                                               30
          INPUT ART PAT1; INCLR TGM;                                      31
          PRINT2 TRC,COMP,TGC,FF;                                         32
STEP5:    INPUT AGT PAT1; INCLR ART;                                      33
          PRINT2 TRC,COMP,TGC,FF;                                         34
STEP6:    INPUT AGT AGT PAT1; PRINT2 FF, COMP;                            35
STEP7:    INPUT ART AGT PAT1; INCLR AGT; PRID2 FF 2(1111111111111111);    36
          PRINT2; PRAC2 COMP [16:16] 2(1111111111111111);                 37
STEP8:    INPUT TGM AGT PAT1; INCLR ART;                                  38
          PRINT2 COMP; PRAC2 COMP [15:15] 2(1111111111110000100);         39
          PRAC2 COMP [14:14] 2(1111111100001111000);                      40
               REPEAT 2                                                   41
STEP10:   INPUT ACO PAT1;                                                 42
          INSET ACOTRDE -1;                                               43
          PRINT2 COMP,FF,TRC,TGC;                                         44
STEP11:   INPUT ACL PAT PAT1;                                             45
          PRINT2 COMP,FF,TRC,TGC;                                         46
END.                                                                      47

CONGRATULATIONS,  NO SOURCE PROGRAM ERRORS WERE DETECTED IN PASS 1.
          THE TOTAL TIME WAS    0 MINUTES 15.4 SECONDS.

          TIME FOR INITIALIZATION:        0 MINUTES   3.2 SECONDS.
          TIME FOR SCANNING:              0 MINUTES   2.8 SECONDS.
          TIME FOR PARSING:               0 MINUTES   5.3 SECONDS.
          TIME FOR SEMANTIC ACTIONS:      0 MINUTES   4.2 SECONDS.
          TIME FOR ERROR RECOVERY:        0 MINUTES   0.0 SECONDS.
          TIME FOR DEBUG PRINTING:        0 MINUTES   0.0 SECONDS.

            47 CARDS READ AT  183 CARDS PER MINUTE.
```

Figure 13. An Example of TESLA Program - Cont.

6. SIMULATOR CONTROLLER

The function of the simulation controller is to translate TESLA object code and to pass input and storage element values to the simulator body, to control the simulation execution, such as program termination and to place the outputs on an output file without editing for the purpose of obtaining fast simulation.

In order to obtain high speed and efficient execution of the simulator, TESLA source statements are decomposed into several simple commands so that any tasks which can be performed within the compiler are performed there, rather than in the controller.

The controller accepts four input commands, three storage element commands and two miscellaneous commands. Obviously output commands do not affect the controller, so another file having only output commands is generated by the compiler. This is not used for the controller but rather by the separate results display program.

The basic function of input commands is to set the specified array elements to be set to a logical one, logical zero or to specified patterns. These input patterns are masked by bits which are generated from the case limits at the beginning of each one-clock simulation.

Storage element commands cause the storage elements to assume their normal next state, to retain their present states, or to set the states given by associated bit patterns.

Miscellaneous commands consist of End of Input and End of TESLA Program. The End of Input is used at the end of each test step to initiate simulation, i.e., whenever the controller recognizes the End of Input command,

it jumps to the simulator body to execute one clock time of the simulation. The simulated data is placed on an output file for printing. The End of TESLA Program command is used to indicate the end of the simulation. When controller recognizes this, it stops the execution and terminates the program in an orderly manner. A typical format of TESLA's object comman is shown in Figure 14, in which INl means the command name, n is the number of pairs of words where SYMBi presents input interface and signal array subscript and Datai presents the data which is put to the subscript. The detailed description of those commands is also described in [2].

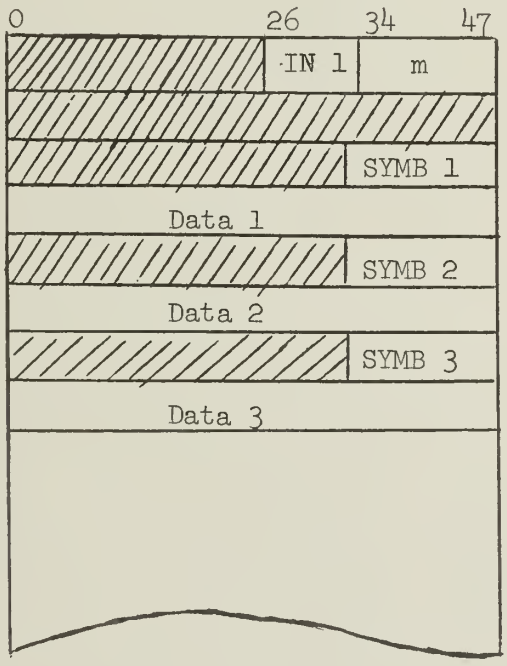


Figure 14.

A Sample Format of TESLA Object Command

7. IMPLEMENTATION

The set of programs for this system is divided into three major groups: the TESLA compiler, the simulator generator group and the results display program. The whole program sequence is shown in Figure 15.

7.1 TESLA Compiler

In order to compile a TESLA program, two programs must be executed; TESLA/MERGE and TESLA/DISK.

a) TESLA/MERGE reads a list of all signal names and their associated array subscripts which has been generated by the simulator generator programs and forms a table of signal names and array subscripts including storage element subscripts.

b) TESLA/DISK is the compiler itself. Referring to the output of TESLA/MERGE, each signal name in a source program is verified and if all signals in the program are valid, the source code is compiled and two object files are created; one contains the simulator control code which specifies all control information and signal values used in the simulator and the other contains the print control object code which specifies the format of printout of the results display program.

7.2 Simulator Generator Group

This group of programs consists of seven subprograms and generates a simulator for each CU or PE printed circuit board automatically. Each subprogram has the facility to call the next program without manual intervention

so that all of these programs are executed automatically in correct order. Each subprogram also generates a history file for tracing the history of the simulator. The ordering program is not currently incorporated in this group because the number of looped packages in the printed circuit boards is small enough to not affect the execution time of the simulation. Each subprogram of this group has the following functions:

a) CUBD/UPDATE accepts an original netlist file and, if necessary, the netlist is updated with correction cards. A reformatted, updated netlist is produced. A sample of update listing is shown in Appendix A. On the left hand side of the listing, four records are shown as inserted and on the right, four records are deleted.

b) CUBD/WEDTR1 - the main job of this program is a signal name sort and assignment of an array element to each signal name. This program also checks for simple wirelist errors, such as signals having no source or having no destination, or having more than eight fan-outs. A part of this signal sort listing is shown in Appendix B. The symbol names in the listing correspond to the array subscripts which are used in the simulator. The A array contains all input interface signals, B contains the output interface signals and C and D are for internal signals. Each array has up to 512 elements.

c) CUBD/WEDTR2 accepts the signal sorted file and sorts by package location and pin. In the process of sorting, an arc-list which describes the directed graph of the board is also generated. In this program, storage elements and connectors are divided into two nodes and each package location is converted to a unique numeric value for efficient execution of the level assignment. Appendix C-1 shows a

part of the listing of a package and pin sort. In it, the four characters at the extreme left show a package location, with the associated package type in parentheses. The three numeric characters to the right of each signal name show the pin to which that signal is connected. A source package and pin is shown in parentheses if a pin of the package is a load, otherwise blanks are put in this position. Appendix C-2 shows the arc-list corresponding to Appendix C-1. Appendix C-3 shows numeric value for each package location and its package type. Note that each flip-flop (type DIL008) has been split in two.

d) CUBD/LEVELER - the arc-list is converted to a source-destination list as shown in Appendix D-1. In this, the integer in parentheses right to the package location is the symbolic name of a package and integers in the source or destination row correspond to other package locations. Using this list, the level assignment procedure assigns a level to each package in ascending and descending order until no more packages can be assigned. The output listing of this procedure is shown in Appendix D-2. The flag bit is set if a package has been assigned a level. Therefore, if both the FWD and BKWD flag bits are zero, the corresponding package is a member of a loop or bounded by loops. In this example, A014, A015, etc. as unassigned.

e) CUBD/REDUCE - the first part of this program prints a listing of assigned packages and unassigned packages and generates the arc-list of only the unassigned packages as shown in Appendix E-1. A loop detection procedure separates each loop and converts each loop to one pseudo package. Finally, a source-destination list using pseudo packages

is produced. In Appendix E-2, the members of each loop are listed. For example, a loop called CHN01 consists of A014 and A015. In this example, there are six loops and one non-looped package (A139) which is bounded by loops. Using these pseudo packages (CHN01 through CHN06 and A139), the source-destination list is printed.

f) CUBD/HSKEEP - using the previous source-destination file, an ascending level assignment is done by the same procedure as in CUBD/LEVELER. This is shown in Appendix F-1. Finally, all packages are reassigned so that the levels of the looped packages starts from level fifty and the last level becomes one hundred, as shown in Appendix F-2.

g) CUBD/SIMGEN - referring to the levels just assigned to the packages and to the package-pin sorted netlist generated by CUBD/WEDTR2, this program generates sets of logic equations whose variables are array elements corresponding to signals in the netlist. For looped packages, a iterative statement and test statement for the stabilization are inserted so that if the outputs of the packages have not stabilized after thirty repetitions, a message is printed out to indicate a possible race condition. Because of the nature of the B-5500 ALGOL compiler, some redundant words such as BEGIN, DEFINE, and END are inserted to force program segmentation since no more than 1022 words of object code may be in any segment. In addition to generating the simulator body, this program creates a small parameter file which contains the number of input, output, internal and storage signals and the name of a board to be simulated, etc. Also, all signal names are printed, partitioned into their respective input, output, and internal arrays signal as shown in Appendix G-2.

In Appendix G-1, a part of simulator body is shown. In this, VTRUE stands for logical one, FALSE for logical zero and OEMPTY for any empty output pin. For looped packages (for example, level 53, A137), a WHILE statement is used for the iteration. The Y array stores the current values of outputs and Z array stores the previous output values. After the evaluation of each set of logic equations, the elements of the Y and Z arrays are compared. If the comparison disagrees, STABLE will be set to false and the evaluation will be repeated.

h) CUBD/MERGE - reads the parameter and simulator body file and the head and tail (which were written manually) and combines them into a complete simulator program. The head and tail program is common to all boards.

7.3 Simulator

<BOARD NAME>/SIMULA is generated by the above simulator generator group. The simulator accepts the simulation input file which is created by the TESLA compiler and generates a file which contains necessary information for the results display, including the simulator history. During a simulation, if there are no errors, only step numbers and the time used is printed (Appendix H). The print-out of the simulation results is done by the Results Display program.

7.4 Result Display Program

CUBD/PRINTER reads a print control file generated by TESLA compiler and makes a listing of each test case in the requested format. A sample listing is shown in Appendix I.

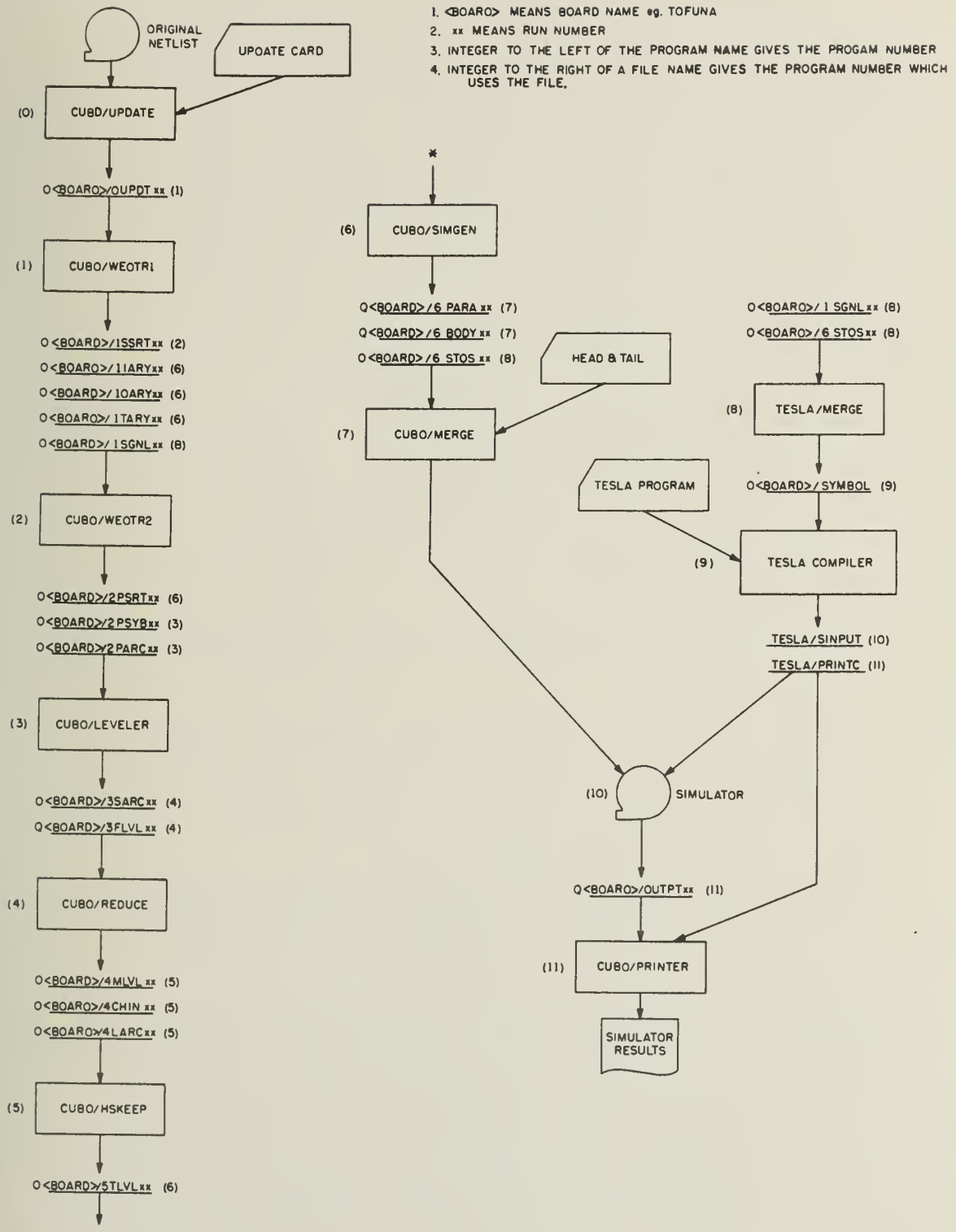


Figure 15. Program System of Simulator.

8. CONCLUSION

The description of this paper focused on what was originally to have been the ILLIAC IV Control Unit printed circuit board simulator system. This system is quite general, however, and was quickly and easily modified to also produce Processing Element printed circuit board simulators and the fault test pattern generation system for both CU and PE printed circuit boards. Both CU section simulators and PE simulators are being developed based on most of those programs described here. This simulator system is extensively used for test pattern generation, in which the simulator body generator (CUBD/SIMGEN) is modified so that it can generate sets of failure inserted logical equations, as well as correct logical equations. The detailed method of test pattern generation is described in [9].

The differences between PE printed circuit boards and CU printed circuit boards are as follows:

	PE	CU
Number of IC packages (max)	20	165
Number of interface pins (max)	100	500
Logical value representation	Positive voltage = Logic "1"	Negative voltage = Logic "1"

From a programmatic point of view, the CU board simulator system can treat PE boards since the size of a PE board is much smaller than that of a CU board. However, because the logical value representations are different (even though the same IC's are used), there are two different simulator body generators. In the case of PE generator, AND and OR elements in the CU representation are changed to OR and AND elements and logical zeroes and ones in CU representation

are changed to logical ones and zeroes. The PE board simulator body generator is called CUBD/PESIM and has been operational for almost a year.

The simulator has been used for two purposes; first, for understanding and debugging the logic of boards and second, for verifying manually prepared test patterns written in TESLA for the boards. Both of these uses have been applied to CU boards, particularly the latter since it is sometimes difficult to find test patterns within a reasonable time for some types of logic using the test pattern generation system. The PE board simulator system has not been used extensively for the first purpose since the logic of the PE boards is so simple that there is no difficulty in understanding and to debugging them. It has, however, proven quite useful for correct simple, but basic errors in manually written netlists by simply looking at the error listings which are produced while generating the simulator. The PE board test generation system has been very successively used to quickly produce diagnostics for all 35 PE board types.

Although the CU board simulator has been used for debugging, more extensive use of a simulator will be made when a section simulator is constructed.

Execution time for the generation of a typical CU board simulator is less than ten minutes; for a typical PE board simulator, it is less than five minutes. The execution time for a typical CU board simulator itself is about fifty miliseconds per clock per set of up to 47 test patterns.

It has also been proven that this method of generating simulators is quite general, since almost all of the programs can be used for various simulators, such as, PE and CU boards, CU sections, and the whole PE, and test pattern generation for each of them.

APPENDIX A

A SAMPLE UPDATE LISTING OF CUBD/UPDATE

```
                                                ILLIAC IV BDA D NA E=T FUNL                    DELETED,TAPE ERROR

                                                T1CLK00FND BUFF 008 CLOCKS S TDFUND            DELETED
                                                T1CLK01FND BUFF 007 CLOCKS S TDFUND            DELETED
                                                T1CLK01FND A079 013 DIL022 L TDFUND            DELETED
 T=CLK===HI A082 002 DIL013 S TDFUND            INSERTED
 T=CLK===HI BUFF 016 CLKBUF L TDFUND            INSERTED
 T1CLK00FND BUFF 008 CLKBUF S TDFUND            INSERTED
 T1CLK01FND BUFF 007 CLKBUF S TDFUND            INSERTED
                            ORIGINAL NUMBER OF RECORDS= 972
                            NUMBER OF RECORDS IN THE UPDATED FILE= 972


                            NUMBER OF INSERT CARDS =  2

                            NUMBER OF DELETE CARDS=  1

                            NUMBER OF CHANGE CARDS=  2

TIME USED TO OBTAIN THIS RESULT:

            PROCESSOR TIME:     0 MINUTES 29 SECONDS,
            I/O TIME:           0 MINUTES 29 SECONDS,
            IN               1323 MINUTES 22 SECONDS.

DATE:            MONDAY,     1/ 5/70,   10: 3 PM.
```

APPENDIX B

A SAMPLE SIGNAL SORT LISTING OF CUBD/WEDTR1

REMARK	SIG NAME	PKGE ID	PIN	PKGE TYPE	S/L	SYMBOL NAME	SEQ	
	CLK----BC1	BUFF	016	CLKBUF	L	C[0]	1	1
	CLK----BC1	A200	002	DIL013	S	C[0]	1	2
	CLK-----C0	A200	016	DIL013	L	A[0]	2	3
	CLK-----C0	P-02	A03	PIN	S	A[0]	2	4
	CLK-----C1	A200	014	DIL013	L	A[1]	3	5
	CLK-----C1	P-02	A07	PIN	S	A[1]	3	6
	GROUND--10	A024	003	DILDLY	L	C[1]	4	7
	GROUND--10	A024	001	DILDLY	S	C[1]	4	8
	GROUND--11	A025	006	DILDLY	L	C[2]	5	9
	GROUND--11	A025	007	DILDLY	S	C[2]	5	10
	GROUND--12	A025	003	DILDLY	L	C[3]	6	11
	GROUND--12	A025	001	DILDLY	S	C[3]	6	12
	GROUND--13	A027	006	DILDLY	L	C[4]	7	13
	GROUND--13	A027	007	DILDLY	S	C[4]	7	14
	GROUND--14	A027	003	DILDLY	L	C[5]	8	15
	GROUND--14	A027	001	DILDLY	S	C[5]	8	16
	GROUND--15	A028	006	DILDLY	L	C[6]	9	17
	GROUND--15	A028	007	DILDLY	S	C[6]	9	18
	GROUND--16	A028	003	DILDLY	L	C[7]	10	19
	GROUND--16	A028	001	DILDLY	S	C[7]	10	20
	GROUND--17	A036	006	DILDLY	L	C[8]	11	21
	GROUND--17	A036	007	DILDLY	S	C[8]	11	22
	GROUND--18	A036	003	DILDLY	L	C[9]	12	23
	GROUND--18	A036	001	DILDLY	S	C[9]	12	24
	GROUND--19	A046	006	DILDLY	L	C[10]	13	25
	GROUND--19	A046	007	DILDLY	S	C[10]	13	26
	GROUND--20	A046	003	DILDLY	L	C[11]	14	27
	GROUND--20	A046	001	DILDLY	S	C[11]	14	28
	GROUND--21	A056	006	DILDLY	L	C[12]	15	29
	GROUND--21	A056	007	DILDLY	S	C[12]	15	30

TFR-5D--R0	P-02	B28	PIN	L	R[28]	75	158
TFR-5D--R0	A037	005	DIL027	S	R[28]	75	159
TFR-5D--R1	P-02	B20	PIN	L	R[29]	76	160
TFR-5D--R1	A037	004	DIL027	S	R[29]	76	161
T-ALT---P0	A139	014	DIL011	L	C[36]	77	162
T-ALT---P0	A139	009	DIL011	L	C[36]	77	163
T-ALT---R0	A014	014	DIL011	L	C[36]	77	164
T-ALT---R0	A014	009	DIL011	L	C[36]	77	165
T-ALT---R0	A015	004	DIL022	S	C[36]	77	166
T-ALT---R1	A139	012	DIL011	L	C[37]	78	167
T-ALT---R1	A014	012	DIL011	L	C[37]	78	168
T-ALT---R1	A015	002	DIL022	S	C[37]	78	169
T-ALT---R0	A129	013	DILOUC	L	C[38]	79	170
T-ALT---R0	A129	012	DILOUC	L	C[38]	79	171
T-ALT---R0	A126	013	DILOUC	L	C[38]	79	172
T-ALT---R0	A126	012	DILOUC	L	C[38]	79	173
T-ALT---R0	A121	013	DILOUC	L	C[38]	79	174
T-ALT---R0	A121	012	DILOUC	L	C[38]	79	175
T-ALT---R0	A122	004	DILODC	S	C[38]	79	176
T-ALT---R1	A128	013	DILOUC	L	C[39]	80	177
T-ALT---R1	A128	012	DILOUC	L	C[39]	80	178
T-ALT---R1	A125	013	DILOUC	L	C[39]	80	179
T-ALT---R1	A125	012	DILOUC	L	C[39]	80	180
T-ALT---R1	A117	013	DILOUC	L	C[39]	80	181
T-ALT---R1	A117	012	DILOUC	L	C[39]	80	182
T-ALT---R1	A015	001	DIL022	L	C[39]	80	183
T-ALT---R1	A122	002	DILODC	S	C[39]	80	184
T-ALT---S1	A122	014	DILODC	L	A[10]	81	185
T-ALT---S1	P-01	B32	PIN	S	A[10]	81	186
T-BINDATA1	A081	013	DIL004	L	C[40]	82	187
T-BINDATA1	A076	002	DIL022	S	C[40]	82	188
T-BIT35-G1	A130	012	DIL004	L	C[41]	83	189
T-BIT35-G1	A130	013	DIL004	L	C[41]	83	190
T-BIT35-G1	A076	008	DIL022	L	C[41]	83	191
T-BIT35-G1	A132	007	DIL022	S	C[41]	83	192
T-BITF0-G1	A091	011	DILDLY	L	C[42]	84	193
T-BITF0-G1	A090	008	DIL007	S	C[42]	84	194
T-BITF0-L0	A132	013	DIL022	L	C[43]	85	195
T-BITF0-L0	A085	008	DIL008	S	C[43]	85	196

APPENDIX C. LISTINGS OF CUBD/WEDTR2

C-1

A SAMPLE PACKAGE NAME AND PIN SORT LISTING OF CUBD/WEDTR2

```
   CU BOARD=TD1SP

A001      (D1L022)          1
     001   T-NVOLT--1   (D1     004)      002   T-TFC--KG1   (          )      004   T-TFC--KG0   (          )
     005   T-TFC--KG3   (          )      007   T-TFC--KG2   (          )      008   T-INSFTEG1   (A008  005)
     009   T-NVOLT--1   (D1     004)      011   T-NVOLT--1   (D1     004)      012   T-CLK02--1   (BUFF  003)
     013   T-CLK02--1   (BUFF   003)      014   T-INSFTEG1   (A009  005)      016   T-NVOLT--1   (D1     004)


A002      (D1L008)          2
     002   T-TFC04-L1   (          )      007   T-TFC05-L1   (          )      011   T-TFC--KG0   (A001  004)
     012   T-TD----G1   (A014   005)      013   T-TFC--KG1   (A001   002)      014   T-TFC05-D1   (A007  013)


A003      (D1L008)          3
     002   T-TFC02-L1   (          )      007   T-TFC03-L1   (          )      011   T-TFC--KG0   (A001  004)
     012   T-TFC04-D1   (A007   016)      013   T-TFC--KG1   (A001   002)      014   T-TFC03-01   (A008  013)


A004      (D1L008)          4
     002   T-TFC00-L1   (          )      007   T-TFC01-L1   (          )      011   T-TFC--KG2   (A001  007)
     012   T-TFC02-D1   (A008   016)      013   T-TFC--KG3   (A001   005)      014   T-TFC01-D1   (A010  013)


A005      (D1L008)          5
     002   T-TDR06-L1   (          )      007   T-TOR07-L1   (          )      011   T-TFC--KG2   (A001  007)
     012   T-TFC00-D1   (A010   016)      013   T-TFC--KG3   (A001   005)      014   T-TOR07-D1   (A011  013)


A006      (D1L007)          6
     001   T-TFC02-G1   (          )      004   T-TFC03-G1   (          )      005   T-TFC04-G1   (          )
     008   T-TFC05-G1   (          )      009   T-TFC05-L1   (A002   007)      012   T-TFC04-L1   (A002  002)
     013   T-TFC03-L1   (A003   007)      014   T-NVOLT--1   (D1     004)      016   T-TFC02-L1   (A003  002)


A007      (F1L01Y)          7
     011   T-TFC05-G1   (A006   008)      013   T-TFC05-01   (          )      014   T-TFC04-G1   (A006  005)
     016   T-TFC04-D1   (          )

******** GROUND SIGNALS ARE DELETED. ********


A008      (D1L01Y)          8
     011   T-TFC03-G1   (A006   004)      013   T-TFC03-D1   (          )      014   T-TFC02-G1   (A006  001)
     016   T-TFC02-D1   (          )

******** GROUND SIGNALS ARE DELETED. ********


A009      (D1L007)          9
     001   T-TDR06-G1   (          )      004   T-TDR07-G1   (          )      005   T-TFC00-G1   (          )
     008   T-TFC01-G1   (          )      009   T-TFC01-L1   (A007   007)      012   T-TFL00-L1   (A004  002)
     013   T-TDR07-L1   (A005   007)      014   T-NVOLT--1   (D1     004)      016   T-TDR06-L1   (A005  002)
```

```
A010      (FILDLY)          10
     011    T-TFC01-G1   (A009  008)       013   T-TFC01-D1   (          )       014   T-TFC00-G1   (A009  005)
     016    T-TFC00-D1   (          )

******** GROUND SIGNALS ARE DELETED. ********


A011      (FILDLY)          11
     011    T-TDR07-G1   (A009  004)       013   T-TDR07-D1   (          )       014   T-TDR06-G1   (A009  001)
     016    T-TDR06-D1   (          )

******** GROUND SIGNALS ARE DELETED. ********


A012      (D1LODC)          12
     004    T-TD-1--R1   (          )       007   T-TD-2--R1   (          )       011   T-NVOLT--1   (D1    004)
     C12    TD-2------   (P-03  B46)       014   TD-1------   (P-03  B20)       016   T-NVOLT--1   (D1    004)


A013      (D1LODC)          13
     004    T-RQ-1--R1   (          )       007   T-RQ-2--R1   (          )       011   T-NVOLT--1   (D1    004)
     012    RQ-2------   (P-03  B28)       014   RQ-1------   (P-03  B04)       016   T-NVOLT--1   (D1    004)


A014      (FILO11)          14
     004    T-RQ----G1   (          )       005   T-TD----G1   (          )       007   T-TD----GO   (          )
     008    T-TD-1--R1   (A012  004)       009   T-ALT---B0   (A015  004)       011   T-TD-2--R1   (A012  007)
     012    T-ALT---R1   (A015  002)       013   T-RQ-2--R1   (A013  007)       014   T-ALT---B0   (A015  004)
     016    T-RQ-1--R1   (A013  004)


A015      (FILO22)          15
     001    T-ALT---R1   (A122  002)       002   T-ALT---R1   (          )       004   T-ALT---B0   (          )
     005    T-DSPDPSG1   (          )       008   T-RQ----G1   (A014  004)       009   T-TA----L0   (A120  008)
     011    T-RA----L0   (A116  008)       012   T-NVOLT--1   (D1    004)       013   T-NVOLT--1   (D1    004)
     014    T-NVOLT--1   (D1    004)       016   T-NVOLT--1   (D1    004)


A016      (D1LO08)          16
     002    T-DSPDP-L3   (          )       007   T-DSPDP-L1   (          )       011   T-DSPDPKG0   (A152  007)
     012    T-DSPDPSG1   (A015  005)       013   T-DSPDPKG1   (A152  005)       014   T-DSPDPSG1   (A015  005)


A017      (FILO11)          17
     002    T-TRG01-D0   (          )       004   T-TRG01-D1   (          )       005   T-DSPDPEG1   (          )
     008    T-DSPDPCG1   (A137  002)       009   T-NVOLT--1   (D1    004)       011   T-DSPDPSG1   (A015  005)
     012    T-NVOLT--1   (D1    004)       013   T-PVOLT--0   (D1    005)       014   T-NVOLT--1   (D1    004)
     016    T-TRG01-D3   (A094  (16)


A018      (FILO22)          18
     001    T-NVOLT--1   (D1    004)       002   T-TDR--KG1   (          )       004   T-TDR--KGO   (          )
     005    T-TDR--KG3   (          )       007   T-TDR--KG2   (          )       008   T-INSFTEG1   (A098  005)
     009    T-NVOLT--1   (D1    004)       011   T-NVOLT--1   (D1    004)       012   T-CLK01--1   (BUFF  002)
     C13    T-CLK01--1   (BUFF  002)       014   T-INSFTEG1   (A098  005)       016   T-NVOLT--1   (D1    004)


A019      (FILOOP)          19
```

APPENDIX C. LISTINGS OF CUBD/WEDTR2

C-2

A SAMPLE ARC LISTING OF CUBD/WEDTR2

CU BOARD=TDISP ARC LIST

```
A001 ← DI        A001 ← A098      A001 ← BUFF
A002 ← A001      A002 ← A014      A002 ← A007
A003 ← A001      A003 ← A007      A003 ← A008
A004 ← A001      A004 ← A008      A004 ← A010
A005 ← A001      A005 ← A010      A005 ← A011
A006 ← A002      A006 ← A003      A006 ← DI
A007 ← A006
A008 ← A006
A009 ← A004      A009 ← A005      A009 ← DI
A010 ← A009
A011 ← A009
A012 ← DI        A012 ← P=03
A013 ← DI        A013 ← P=03
A014 ← A012      A014 ← A015      A014 ← A013
A015 ← A122      A015 ← A014      A015 ← A120      A015 ← A116      A015 ← DI
A016 ← A152      A016 ← A015
A017 ← A137      A017 ← DI        A017 ← A015      A017 ← A094
A018 ← DI        A018 ← A098      A018 ← BUFF
A019 ← A018      A019 ← A011      A019 ← A024
A020 ← A018      A020 ← A024      A020 ← A025
A021 ← A018      A021 ← A025      A021 ← A027
A022 ← A018      A022 ← A027      A022 ← A028
A023 ← A020      A023 ← A019      A023 ← DI
A024 ← A023
A025 ← A023
A026 ← A021      A026 ← A022      A026 ← DI
A027 ← A026
A028 ← A026
A029 ← DI        A029 ← A098      A029 ← BUFF
A030 ← A019      A030 ← A005      A030 ← DI
A031 ← A020      A031 ← A021      A031 ← DI
A032 ← A029      A032 ← A028      A032 ← DI
A033 ← A116      A033 ← A139      A033 ← A103      A033 ← A131
A034 ← A002      A034 ← A033      A034 ← A041
A035 ← A039      A035 ← A034
A036 ← A035      A036 ← A040
A037 ← A035      A037 ← DI
A038 ← DI        A038 ← A033      A038 ← A041
A039 ← DI        A039 ← A038      A039 ← BUFF
A040 ← A039      A040 ← A034
A041 ← DI        A041 ← A036
A042 ← A040      A042 ← DI
A043 ← DI        A043 ← A041      A043 ← A051
A044 ← A003      A044 ← A033      A044 ← A051
A045 ← A049      A045 ← A044
A046 ← A045      A046 ← A050
A047 ← A045      A047 ← DI
A048 ← DI        A048 ← A033      A048 ← A043
A049 ← DI        A049 ← A048      A049 ← BUFF
A050 ← A049      A050 ← A044
A051 ← A046      A051 ← DI
A052 ← A045      A052 ← A040      A052 ← A035      A052 ← DI        A052 ← A135
A053 ← DI        A053 ← A041      A053 ← A051      A053 ← A053      A053 ← A057
A054 ← A004      A054 ← A033      A054 ← A057
A055 ← A059      A055 ← A054
A056 ← A055      A056 ← A060
A057 ← DI        A057 ← A056
A058 ← DI        A058 ← A033      A058 ← A053
A059 ← DI        A059 ← A058      A059 ← BUFF
```

APPENDIX C. LISTINGS OF CUBD/WEDTR2

C-3

A SAMPLE PACKAGE NAME LISTING OF CUBD/WEDTR2

CU BOARD=TDISP		PACKAGE LIST
0	A001	DIL022
1	A002I	DIL008
2	A002D	DIL008
3	A003I	DIL008
4	A003D	DIL008
5	A004I	DIL008
6	A004D	DIL008
7	A005I	DIL008
8	A005D	DIL008
9	A006	DIL007
10	A007	DILDLY
11	A008	DILDLY
12	A009	DIL007
13	A010	DILDLY
14	A011	DILDLY
15	A012	DIL0DC
16	A013	DIL0DC
17	A014	DIL011
18	A015	DIL022
19	A016I	DIL008
20	A016D	DIL008
21	A017	DIL011
22	A018	DIL022
23	A019I	DIL008
24	A019D	DIL008
25	A020I	DIL008
26	A020D	DIL008
27	A021I	DIL008
28	A021D	DIL008
29	A022I	DIL008
30	A022D	DIL008
31	A023	DIL007
32	A024	DILDLY
33	A025	DILDLY
34	A026	DIL007
35	A027	DILDLY
36	A028	DILDLY
37	A029	DIL022
38	A030	DIL007
39	A031	DIL007
40	A032I	DIL008
41	A032D	DIL008
42	A033	DIL027
43	A034	DIL011
44	A035I	DIL008
45	A035D	DIL008
46	A036	DILDLY
47	A037	DIL027
48	A038	DIL011
49	A039	DIL022
50	A040I	DIL008
51	A040D	DIL008
52	A041	DIL022
53	A042	DIL027
54	A043	DIL022
55	A044	DIL011
56	A045I	DIL008
57	A045D	DIL008
58	A046	DILDLY

APPENDIX D. LISTINGS OF CUBD/LEVELER

D-1

A SAMPLE SOURCE DESTINATION LISTING OF CUBD/LEVELER

SOURCE DESTINATION LIST CU BOARD=TDISP

```
PACKAGE=A001   ( 0)  NO OF SOURCE=  3  NO OF DESTINATION=  4
  SOURCE       :   176    117    175
  DESTINATION  :     1      3      5      7

PACKAGE=A002T  ( 1)  NO OF SOURCE=  3  NO OF DESTINATION=  0
  SOURCE       :     0     17     10
  DESTINATION  :

PACKAGE=A0020  ( 2)  NO OF SOURCE=  0  NO OF DESTINATION=  4
  SOURCE       :
  DESTINATION  :     9     43     95     98

PACKAGE=A003I  ( 3)  NO OF SOURCE=  3  NO OF DESTINATION=  0
  SOURCE       :     0     10     11
  DESTINATION  :

PACKAGE=A0030  ( 4)  NO OF SOURCE=  0  NO OF DESTINATION=  3
  SOURCE       :
  DESTINATION  :     9     55     95

PACKAGE=A004T  ( 5)  NO OF SOURCE=  3  NO OF DESTINATION=  0
  SOURCE       :     0     11     13
  DESTINATION  :

PACKAGE=A0040  ( 6)  NO OF SOURCE=  0  NO OF DESTINATION=  4
  SOURCE       :
  DESTINATION  :    12     67     90     95

PACKAGE=A005T  ( 7)  NO OF SOURCE=  3  NO OF DESTINATION=  0
  SOURCE       :     0     13     14
  DESTINATION  :

PACKAGE=A0050  ( 8)  NO OF SOURCE=  0  NO OF DESTINATION=  4
  SOURCE       :
  DESTINATION  :    12     38     86     90

PACKAGE=A006   ( 9)  NO OF SOURCE=  3  NO OF DESTINATION=  2
  SOURCE       :     2      4    176
  DESTINATION  :    10     11

PACKAGE=A007   ( 10)  NO OF SOURCE=  1  NO OF DESTINATION=  2
  SOURCE       :     9
  DESTINATION  :     1      3

PACKAGE=A008   ( 11)  NO OF SOURCE=  1  NO OF DESTINATION=  2
  SOURCE       :     9
  DESTINATION  :     3      5
```

APPENDIX D. LISTINGS OF CUBD/LEVELER

D-2

A SAMPLE LEVEL ASSIGNMENT LISTING OF CUBD/LEVELER

LEVEL ASSIGNMENT LIST CU BOARD=TDISP NO OF PACKAGES= 187

PACKAGE	FWD LVL FLAG	FWD LVL	BKWD LVL FLAG	BKWD LVL
A002n	1	0	0	3
A003n	1	0	0	3
A004n	1	0	0	3
A005n	1	0	0	3
A016n	1	0	0	7
A019n	1	0	0	3
A020n	1	0	0	3
A021n	1	0	0	3
A022n	1	0	1	3
A032n	1	0	0	0
A035n	1	0	0	2
A040n	1	0	0	2
A045n	1	0	0	2
A050n	1	0	0	3
A055n	1	0	0	3
A060n	1	0	0	3
A085n	1	0	0	3
A086n	1	0	0	3
A087n	1	0	0	6
A089n	1	0	0	7
A101n	1	0	0	6
A104n	1	0	0	3
A108n	1	0	0	0
A112n	1	0	1	2
A116n	1	0	0	7
A120n	1	0	0	7
A124n	1	0	1	2
A135n	1	0	0	0
BUFF	1	0	0	2
DJ	1	0	0	7
F=01n	1	0	0	0
F=02n	1	0	0	2
F=03n	1	0	0	0
F=04n	1	0	0	0
F=05n	1	0	0	0
A006	1	1	1	2
A009	1	1	1	2
A012	1	1	0	0
A013	1	1	0	0
A023	1	1	1	2
A026	1	1	1	2
A030	1	1	1	1
A031	1	1	1	1
A036	1	1	0	0
A037	1	1	1	1
A042	1	1	1	1
A046	1	1	0	0
A047	1	1	1	1
A052	1	1	0	1
A056	1	1	0	0
A061	1	1	1	2
A063	1	1	0	0
A065	1	1	1	2
A067	1	1	0	0
A103	1	1	0	7
A109	1	1	0	0
A110	1	1	0	0
A122	1	1	0	3

A132	1	1	0	0
A138	1	1	0	0
A141	1	1	0	0
A142	1	1	0	0
A153	1	1	1	1
A155	1	1	1	1
A200	1	1	1	0
A007	1	2	1	1
A008	1	2	1	1
A010	1	2	1	1
A011	1	2	1	1
A024	1	2	1	1
A025	1	2	1	1
A027	1	2	1	1
A028	1	2	1	1
A041	1	2	0	4
A051	1	2	0	4
A057	1	2	0	2
A062	1	2	1	1
A064	1	2	0	1
A066	1	2	1	1
A068	1	2	0	1
A076	1	2	0	0
A117	1	2	1	1
A121	1	2	1	1
A125	1	2	1	1
A126	1	2	1	1
A127	1	2	1	2
A140	1	2	0	0
P=05†	1	2	1	0
A003	1	3	1	3
A069	1	3	0	0
A070	1	3	0	0
A073	1	3	0	0
A074	1	3	0	0
A077	1	3	0	0
A078	1	3	0	3
A079	1	3	0	0
A082	1	3	0	0
A128	1	3	1	1
A129	1	3	1	1
P=02†	1	3	1	0
P=04†	1	3	1	0
A071	1	4	0	0
A075	1	4	0	0
A080	1	4	0	0
A083	1	4	0	0
A090	1	4	1	2
P=03†	1	4	1	0
A072	1	5	0	0
A091	1	5	1	1
A092	1	5	1	1
A014	0	2	0	3
A015	0	2	0	4
A053	0	3	0	3
A081	0	6	0	0
A096	0	1	0	0
A097	0	1	0	3
A098	0	2	0	3
A102	0	2	0	1
A111	0	2	0	2

A114	0	1	0	0
A115	0	2	0	1
A118	0	4	0	0
A119	0	2	0	1
A130	0	2	0	0
A131	0	1	0	4
A133	0	2	0	0
A134	0	1	0	1
A136	0	1	0	3
A137	0	2	0	4
A139	0	2	0	7
A105	0	2	1	6
A093	0	1	1	5
A094	0	0	1	4
A017	0	1	1	3
A033	0	2	1	3
A038	0	3	1	2
A048	0	4	1	2
A058	0	1	1	2
A088	0	2	1	2
A099	0	2	1	2
A106	0	1	1	2
A001	0	1	1	1
A018	0	1	1	1
A029	0	1	1	1
A034	0	3	1	1
A039	0	1	1	1
A044	0	3	1	1
A049	0	1	1	1
A054	0	3	1	1
A059	0	1	1	1
A084	0	1	1	1
A095	0	0	1	1
A100	0	1	1	1
A107	0	1	1	1
A113	0	1	1	1
A123	0	2	1	1
A152	0	1	1	1
A154	0	1	1	1
A002†	0	3	1	0
A003†	0	3	1	0
A004†	0	3	1	0
A005†	0	3	1	0
A016†	0	0	1	0
A019†	0	3	1	0
A020†	0	3	1	0
A021†	0	3	1	0
A022†	0	3	1	0
A032†	0	3	1	0
A035†	0	0	1	0
A040†	0	0	1	0
A045†	0	0	1	0
A050†	0	0	1	0
A055†	0	0	1	0
A060†	0	0	1	0
A085†	0	6	1	0
A086†	0	6	1	0
A087†	0	6	1	0
A089†	0	0	1	0
A101†	0	2	1	0
A104†	0	1	1	0

A108↑	0	0	1	0
A112↑	0	1	1	0
A116↑	0	1	1	0
A120↑	0	1	1	0
A124↑	0	1	1	0
A135↑	0	1	1	0
P-01↑	0	2	1	0

TIME USED TO OBTAIN THIS RESULT:

PROCESSOR TIME: 1 MINUTES 3 SECONDS.
I/O TIME: 0 MINUTES 52 SECONDS.
IN 2 MINUTES 0 SECONDS.

DATE: THURSDAY, 8/ 6/69, 5:16 PM.

APPENDIX E. LISTINGS OF CUBD/REDUCE

E-1

A SAMPLE LEVEL ASSIGNMENT EXCLUDING LOOPED PACKAGES AND ARC LISTINGS OF LOOPED PACKAGES OF CUBD/REDUCE

```
INITIAL LEVEL ASSIGNMENT LIST CU BOARD=TDISP


LEVEL=  0   A002D  A003D  A004D  A005D  A016D  A019D  A020D  A021D  A022D  A032D
            A035D  A040D  A045D  A050D  A055D  A060D  A085D  A086D  A087D  A089D
            A101D  A104D  A108D  A112D  A116D  A120D  A124D  A135D  DI     P=01D
            P=02D  P=03D  P=04D  P=05D
LEVEL=  1   A006   A009   A012   A013   A023   A026   A030   A031   A036   A037
            A042   A046   A047   A052   A056   A061   A063   A065   A067   A103
            A109   A110   A122   A132   A138   A141   A142   A153   A155   A200

LEVEL=  2   A007   A008   A010   A011   A024   A025   A027   A028   A041   A051
            A057   A062   A064   A066   A068   A076   A117   A121   A125   A126
            A127   A140   P1FF
LEVEL=  3   A043   A069   A070   A073   A074   A077   A078   A079   A082   A126
            A129
LEVEL=  4   A071   A075   A080   A083   A09C
LEVEL=  5   A072   A091   A092
LEVEL= 94   A105
LEVEL= 95   A093
LEVEL= 96   A094
LEVEL= 97   A017   A033
LEVEL= 98   A038   A048   A058   A088   A099   A106
LEVEL= 99   A001   A018   A020   A034   A039   A044   A049   A054   A059   A064
            A095   A100   A107   A113   A123   A152   A154
LEVEL=100   P=05I  P=02I  P=04I  P=03I  A002I  A003I  A004I  A005I  A016I  A019I
            A020I  A021I  A022I  A032I  A035I  A040I  A045I  A050I  A055I  A060I
            A085I  A086I  A087I  A089I  A101I  A104I  A108I  A112I  A116I  A120I
            A124I  A135I  P=01I


LOOPED PACKAGE= A014   A015   A053   A081   A096   A097   A098   A102   A111   A114
                A115   A118   A119   A130   A131   A133   A134   A136   A137 A139


LOOPED ARC LIST CU BOARD=TDISP


A015  ← A014
A102  ← A014
A014  ← A015
A139  ← A015
A053  ← A053
A096  ← A081
A097  ← A096
A136  ← A097
A111  ← A098
A098  ← A102
A081  ← A111
A114  ← A111
A133  ← A111
A134  ← A111
A115  ← A114
A114  ← A115
A119  ← A118
A137  ← A118
A118  ← A119
A131  ← A130
```

APPENDIX E. LISTINGS OF CUBD/REDUCE

E-2

A SAMPLE LISTING OF THE MEMBER OF LOOPED PACKAGES AND ITS SOURCE DESTINATION LISTING OF CUBD/REDUCE

```
CHAIN PACKAGE:CHN01   A014   A015   CHN02   A053   CHN03   A081   A096   A097   A098
              A102    A111   A130   A131    A133   A134    A136   CHN04  A114   A115
              CHN05   A118   A119   CHN06   A137

LOOPED PACKAGE:  7    CHN01  CHN02  CHN03  CHN04  CHN05  CHN06  A139

PACKAGE=CHN01   (  0)   NO OF SOURCE=  0   NO OF DEST=  2
SOURCE:
DEST  :     2      6

PACKAGE=CHN02   (  1)   NO OF SOURCE=  1   NO OF DEST=  0
SOURCE:      6
DEST  :

PACKAGE=CHN03   (  2)   NO OF SOURCE=  2   NO OF DEST=  2
SOURCE:      0      6
DEST  :     3      5

PACKAGE=CHN04   (  3)   NO OF SOURCE=  3   NO OF DEST=  0
SOURCE:      2      5      6
DEST  :

PACKAGE=CHN05   (  4)   NO OF SOURCE=  1   NO OF DEST=  1
SOURCE:      6
DEST  :     5

PACKAGE=CHN06   (  5)   NO OF SOURCE=  3   NO OF DEST=  1
SOURCE:      2      4      6
DEST  :     3

PACKAGE=A139    (  6)   NO OF SOURCE=  1   NO OF DEST=  4
SOURCE:      0
DEST  :     2      3      4      5


TIME USED TO OBTAIN THIS RESULT:

          PROCESSOR TIME:    0 MINUTES 34 SECONDS.
          I/O TIME:          0 MINUTES 40 SECONDS.
          IN                 2 MINUTES  8 SECONDS.
```

APPENDIX F. LISTINGS OF CUBD/HSKEEP

F-1

A SAMPLE LEVEL ASSIGNMENT LISTING FOR LOOPED PACKAGES OF CUBD/HSKEEP

```
PACKAGE  CHNO1
SOURCE     NARCS=  0,  NARCD=  0,LVD=1,LVL=  0

DESTINATION NARCS=  2,  NARCD=  2,LVD=1,LVL=  4
                 2          6


PACKAGE  CHNO2
SOURCE     NARCS=  0,  NARCD=  0,LVD=1,LVL=  0

DESTINATION NARCS=  0,  NARCD=  0,LVD=1,LVL=  0


PACKAGE  CHNO3
SOURCE     NARCS=  2,  NARCD=  2,LVD=1,LVL=  2
                 0          6
DESTINATION NARCS=  2,  NARCD=  2,LVD=1,LVL=  2
                 3          5


PACKAGE  CHNO4
SOURCE     NARCS=  3,  NARCD=  3,LVD=1,LVL=  4
                 2          5          6
DESTINATION NARCS=  0,  NARCD=  0,LVD=1,LVL=  0


PACKAGE  CHNO5
SOURCE     NARCS=  1,  NARCD=  1,LVD=1,LVL=  2
                 4
DESTINATION NARCS=  1,  NARCD=  1,LVD=1,LVL=  2
                 5


PACKAGE  CHNO6
SOURCE     NARCS=  3,  NARCD=  3,LVD=1,LVL=  3
                 2          4          6
DESTINATION NARCS=  1,  NARCD=  1,LVD=1,LVL=  1
                 3


PACKAGE  A139
SOURCE     NARCS=  1,  NARCD=  1,LVD=1,LVL=  1
                 0
DESTINATION NARCS=  4,  NARCD=  4,LVD=1,LVL=  3
                 2          3          4          5
```

PACKAGE	FWD LVL FLAG	FWD LVL	BKWD LVL FLAG	BKWD LVL
CHNO1	1	0	1	4
CHNO2	1	0	1	0
A139	1	1	1	3
CHNO3	1	2	1	2
CHNO5	1	2	1	2
CHNO6	1	3	1	1
CHNO4	1	4	1	0

APPENDIX F. LISTINGS OF CUBD/HSKEEP

F-2

A SAMPLE FINAL LEVEL ASSIGNMENT LISTING OF CUBD/HSKEEP

```
FINAL LEVEL ASSIGNMENT LIST      CU BOARD=TDISP

LEVEL=  0   A0020   A0030   A0040   A0050   A0160   A0190   A0200   A0210   A0220   A0320
            A0350   A0400   A0450   A0500   A0550   A0600   A0850   A0860   A0870   A0890
            A1010   A1040   A1080   A1120   A1160   A1200   A1240   A1350   DI      P=010
            P=020   P=030   P=040   P=050
LEVEL=  1   A006    A009    A012    A013    A023    A026    A030    A031    A036    A037
            A042    A046    A047    A052    A056    A061    A063    A065    A067    A103
            A109    A110    A122    A132    A138    A141    A142    A153    A155    A200
LEVEL=  2   A007    A008    A010    A011    A024    A025    A027    A028    A041    A051
            A057    A062    A064    A066    A068    A076    A117    A121    A125    A126
            A127    A140    BUFF
LEVEL=  3   A043    A069    A070    A073    A074    A077    A078    A079    A082    A126
            A129
LEVEL=  4   A071    A075    A080    A083    A090
LEVEL=  5   A072    A091    A092
LEVEL= 50   A014    A015    A053
LEVEL= 51   A139
LEVEL= 52   A081    A096    A097    A098    A102    A111    A130    A131    A133    A134
            A136    A118    A119
LEVEL= 53   A137
LEVEL= 54   A114    A115
LEVEL= 94   A105
LEVEL= 95   A093
LEVEL= 96   A094
LEVEL= 97   A017    A033
LEVEL= 98   A038    A048    A058    A088    A099    A106
LEVEL= 99   A001    A018    A029    A034    A039    A044    A049    A054    A059    A084
            A095    A100    A107    A113    A123    A152    A15A
LEVEL=100   P=051   P=021   P=041   P=031   A0021   A0035   A0041   A0051   A0161   A0191
            A0201   A0211   A0221   A0321   A0351   A0401   A0451   A0501   A0551   A0601
            A0651   A0841   A0871   A0891   A1011   A1041   A1081   A1121   A1161   A1201
            A1241   A1351   P=011

TIME USED TO OBTAIN THIS RESULT:

          PROCESSOR TIME:    0 MINUTES  8 SECONDS.
          I/O TIME:          0 MINUTES 12 SECONDS.
          IN                 [redacted] MINUTES 41 SECONDS.

DATE:           MONDAY,   1/ 5/70,    5:24 PM.
```

APPENDIX G. LISTINGS OF CUBD/SIMGEN

G-1

A SAMPLE SIMULATOR BODY LISTING OF CUBD/SIMGEN

```
  DEMPTY ← NOT(C[182] ← (C[099] AND C[063] AND C[169] AND C[183]));      00567300
  C[192] ← NOT(C[193] ← (C[190] AND C[168] AND C[311] AND C[355]));      00567400
  Y[000]←C[193];Y[001]←C[192];Y[002]←C[182];                             00567500
      FOR J←0 STEP 1 UNTIL   2 DO BEGIN                                  00567600
       TEMP←REAL(NOT Z[13,J] EQV Y[J]);                                  00567700
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00567800
       STABLE←FALSE;  Z[13,J]←Y[J];END;END;                              00567900
%                                                                        00568000
%     COMMENT LEVEL= 52   A136(DIL022);                                  00568100
  C[122] ← NOT(C[123] ← (A[015] AND  VTRUE AND  VTRUE AND  VTRUE));      00568200
  DEMPTY ← NOT(C[171] ← (C[067] AND C[173] AND  VTRUE AND  VTRUE));      00568300
  Y[000]←C[171];Y[001]←C[123];Y[002]←C[122];                             00568400
      FOR J←0 STEP 1 UNTIL   2 DO BEGIN                                  00568500
       TEMP←REAL(NOT Z[14,J] EQV Y[J]);                                  00568600
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00568700
       STABLE←FALSE;  Z[14,J]←Y[J];END;END;                              00568800
%                                                                        00568900
%     COMMENT LEVEL= 52   A118(DIL011);                                  00569000
  DEMPTY ← NOT(C[199] ← (C[203] AND C[131]) OR (C[204] AND  VTRUE));     00569100
  DEMPTY ← NOT(C[346] ← ( VTRUE AND A[014]) OR (C[116] AND C[219]));     00569200
  Y[000]←C[346];Y[001]←C[199];                                           00569300
      FOR J←0 STEP 1 UNTIL   1 DO BEGIN                                  00569400
       TEMP←REAL(NOT Z[15,J] EQV Y[J]);                                  00569500
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00569600
       STABLE←FALSE;  Z[15,J]←Y[J];END;END;                              00569700
%                                                                        00569800
%     COMMENT LEVEL= 52   A119(DIL022);                                  00569900
  C[200] ← NOT(C[201] ← (C[199] AND C[087] AND  VTRUE AND  VTRUE));      00570000
  DEMPTY ← NOT(C[204] ← (C[099] AND C[220] AND C[202] AND C[130]));      00570100
  Y[000]←C[204];Y[001]←C[201];Y[002]←C[200];                             00570200
      FOR J←0 STEP 1 UNTIL   2 DO BEGIN                                  00570300
       TEMP←REAL(NOT Z[16,J] EQV Y[J]);                                  00570400
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00570500
       STABLE←FALSE;  Z[16,J]←Y[J];END;END;                              00570600
      END; IF I GEQ 30 THEN WRITE(PRINTER,TIMEOVER,52);                  00570700
END; BEGIN DEFINE NONE=TRUE#;                                            00570800
      I←0; WHILE (I←I+1) LSS 30 AND NOT STABLE OR I LEQ 2  DO           00570900
      BEGIN STABLE!=TRUE;                                                00571000
%                                                                        00571100
%     COMMENT LEVEL= 53   A137(DIL022);                                  00571200
  C[187] ← NOT(DEMPTY ← (C[192] AND C[094] AND A[021] AND  VTRUE));      00571300
  C[094] ← NOT(C[095] ← (C[131] AND C[220] AND  VTRUE AND C[340]));      00571400
  Y[000]←C[095];Y[001]←C[094];Y[002]←C[187];                             00571500
      FOR J←0 STEP 1 UNTIL   2 DO BEGIN                                  00571600
       TEMP←REAL(NOT Z[17,J] EQV Y[J]);                                  00571700
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00571800
       STABLE←FALSE;  Z[17,J]←Y[J];END;END;                              00571900
      END; IF I GEQ 30 THEN WRITE(PRINTER,TIMEOVER,53);                  00572000
END; BEGIN DEFINE NONE=TRUE#;                                            00572100
      I←0; WHILE (I←I+1) LSS 30 AND NOT STABLE OR I LEQ 2  DO           00572200
      BEGIN STABLE!=TRUE;                                                00572300
%                                                                        00572400
%     COMMENT LEVEL= 54   A114(DIL011);                                  00572500
  DEMPTY ← NOT(C[121] ← (C[354] AND  VTRUE) OR (A[013] AND  VTRUE));     00572600
  DEMPTY ← NOT(C[165] ← ( VTRUE AND C[170]) OR ( VTRUE AND C[095]));     00572700
  Y[000]←C[165];Y[001]←C[121];                                           00572800
      FOR J←0 STEP 1 UNTIL   1 DO BEGIN                                  00572900
       TEMP←REAL(NOT Z[18,J] EQV Y[J]);                                  00573000
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                      00573100
       STABLE←FALSE;  Z[18,J]←Y[J];END;END;                              00573200
%                                                                        00573300
```

```
%     COMMENT LEVEL= 54   A115(DI|022);                                    00573400
  DEMPTY ← NOT(C[170] ← (C[099] AND C[121] AND C[131] AND C[219]));        00573500
  C[166] ← NOT(C[167] ← (C[165] AND  VTRUE AND C[091] AND  VTRUE));        00573600
  Y[000]←C[167];Y[001]←C[166];Y[002]←C[170];                               00573700
      FOR J←0 STEP 1 UNTIL   2 DO BEGIN                                    00573800
       TEMP←REAL(NOT Z[19,J] FQV Y[J]);                                    00573900
       IF TEMP NEQ 0 OR TEMP.[1:8] NEQ 0 THEN BEGIN                        00574000
       STABLE←FALSE;  Z[19,J]←Y[J];END;END;                                00574100
      END; IF I GEQ 30 THEN WRITF(PRINTER,TIMEOVER,54);                    00574200
END; BEGIN DEFINE NONE=TRUE#;                                              00574300
%                                                                          00574400
%     COMMENT LEVEL= 94   A105(DI|022);                                    00574500
  C[145] ← NOT(C[146] ← (C[099] AND C[202] AND C[168] AND  VTRUE));        00574600
  DEMPTY ← NOT(C[150] ← (C[068] AND C[130] AND C[219] AND  VTRUE));        00574700
%                                                                          00574800
%     COMMENT LEVEL= 95   A093(DI|007);                                    00574900
  C[058] ←               C[060] AND C[145]  ;                              00575000
  C[062] ←               C[064] AND C[145]  ;                              00575100
  C[066] ←               C[068] AND C[145]  ;                              00575200
  C[211] ←               C[213] AND C[145]  ;                              00575300
%                                                                          00575400
%     COMMENT LEVEL= 96   A094(DI|DIY);                                    00575500
  C[070] ←     C[066];                                                     00575600
  C[210] ←     C[211];                                                     00575700
%                                                                          00575800
%     COMMENT LEVEL= 97   A017(DI|011);                                    00575900
  DEMPTY ← NOT(C[096] ← (C[095] AND  VTRUE) OR (C[101] AND  VTRUE));       00576000
  C[208] ← NOT(C[209] ← ( VTRUE AND  FALSE) OR ( VTRUE AND C[210]));       00576100
%                                                                          00576200
%     COMMENT LEVEL= 97   A033(DI|027);                                    00576300
  DEMPTY ← NOT(C[033] ←  C[034] ←  C[035]   ←  C[168] AND C[131]);         00576400
  DEMPTY ← NOT(C[347] ←  C[348] ←  C[349]   ←  C[219] AND C[184]);         00576500
%                                                                          00576600
%     COMMENT LEVEL= 98   A038(DI|011);                                    00576700
  DEMPTY ← NOT(C[345] ← ( VTRUE AND C[033]) OR ( VTRUE AND C[347]));;      00576800
  DEMPTY ← NOT(C[344] ← (C[347] AND C[329]) OR (C[0331 AND  VTRUE));       00576900
%                                                                          00577000
%     COMMENT LEVEL= 98   A048(DI|011);                                    00577100
  DEMPTY ← NOT(C[342] ← ( VTRUE AND C[034]) OR (C[341] AND C[348]));       00577200
  DEMPTY ← NOT(C[340] ← (C[348] AND C[343]) OR (C[034] AND  VTRUE));       00577300
%                                                                          00577400
%     COMMENT LEVEL= 98   A058(DI|011);                                    00577500
  DEMPTY ← NOT(C[339] ← ( VTRUE AND C[0351) OR (C[336] AND C[349]));       00577600
  DEMPTY ← NOT(C[337] ← (C[349] AND C[338]) OR (C[035] AND  VTRUE));       00577700
%                                                                          00577800
%     COMMENT LEVEL= 98   A088(DI|011);                                    00577900
  DEMPTY ← NOT(C[045] ← (C[070] AND  VTRUE) OR (C[146] AND  VTRUE));       00578000
  DEMPTY ← NOT(C[074] ← ( VTRUE AND C[124]) OR (C[2201 AND C[072]));       00578100
%                                                                          00578200
%     COMMENT LEVEL= 98   A099(DI|011);                                    00578300
  DEMPTY ← NOT(C[205] ← (C[2211 AND C[2091) OR (C[2221 AND C[208]));       00578400
  DEMPTY ← NOT(C[210] ← (C[208] AND C[218]) OR (C[2091 AND C[217]));       00578500
%                                                                          00578600
%     COMMENT LEVEL= 98   A106(DI|004);                                    00578700
  DEMPTY ← NOT(C[152] ← (C[146] AND  VTRUE) OR (C[1501 AND  VTRUE) OR      00578800
                        (C[124] AND C[2541) OR (C[0721 AND C[171]));;      00578900
%                                                                          00579000
%     COMMENT LEVEL= 99   A001(DI|022);                                    00579100
  C[275] ← NOT(C[276] ← (C[124] AND  VTRUE AND  VTRUE AND C[087]));        00579200
  C[273] ← NOT(C[274] ← (C[087] AND C[124] AND  VTRUE AND  VTRUE));        00579300
%                                                                          00579400
```

APPENDIX G. LISTINGS OF CUBD/SIMGEN

G-2

A SAMPLE TOTAL SIGNAL NAME LISTING OF CUBD/SIMGEN

INPUT SIGNAL NAME LIST OF BOARD=TD1SF ARRAY NAME=A(XXX)

	0	1	2	3	4	5	6	7	8	9
0	CLK-----C0	CLK-----C1	LED-1-----	LED-2-----	LOAD-1----	LOAD-2----	RG-1------	RG-2------	TD-1------	TD-2------
10	T-ALT---S1	T-FCL---L1	T-ENREX-G1	T-FURY-C3	T-INIT-CG1	T-INIT--L1	T-NFUL--L1	T-NSS---L1	T-PAREHCS1	T-PHASS-G3
20	T-PHASA-G3	T-REFFD-L0	T-RFT---F1	T-SRLH--G1	T-TR(---S1	T-TR158-L3	T-TR159-L3	T-TR160-L3	T-TR161-F3	T-TR162-F3
30	T-TR163-F3	T-VIEW--S1								

NUMBER OF INPUT SIGNALS= 32

OUTPUT SIGNAL NAME LIST OF BOARD=TD1SF ARRAY NAME=B(XXX)

	0	1	2	3	4	5	6	7	8	9
0	LED-1-----	LED-2-----	RA-1------	RA-2------	RD-1------	RD-2------	RS-1------	RS-2------	TA-1------	TA-2------
10	TFR-3A--F0	TFR-3A--F1	TFR-3B--F0	TFR-3B--F1	TFR-3C--F0	TFR-3C--F1	TFR-3D--F0	TFR-3D--F1	TFR-4A--F1	TFR-4B--F1
20	TFR-4C--F1	TFR-4D--F1	TFR-5A--F0	TFR-5A--F1	TFR-5B--F0	TFR-5B--F1	TFR-5C--F0	TFR-5C--F1	TFR-5D--F0	TFR-5D--F1
30	T-F11FA-L2	T-FCL---L1	T-INIT--F1	T-LED---D1	T-MSHA--D0	T-MSHA--D2	T-MSHA--D4	T-MSHA--D6	T-MSHB--D0	T-MSHB--D2
40	T-MSHB--D4	T-MSHB--D6	T-MSHC--D1	T-MSHC--D3	T-MSHC--D5	T-MSHC--D7	T-MSHD--D1	T-MSHD--D3	T-MSHD--D5	T-MSHD--D7
50	T-NFUL--D1	T-PAREP-D1	T-PHASB-D1	T-PHASA-D1	T-RFT---D1	T-SDDR--G1	T-TDR00-F1	T-TDR01-F1	T-TDR02-L1	T-TDR03-F1
60	T-TDR04-F1	T-TDR05-F1	T-TDR06-F1	T-TDR07-F1	TS-1------	TS-2------				

NUMBER OF OUTPUT SIGNALS= 66

INTERNAL SIGNAL NAME LIST OF BOARD=TD1SF ARRAY NAME=C(XXX)

1	T-TDR04-F1	T-TDR05-F1	T-TDR06-F1	T-TDR07-F1	TS-1------	TS-2------	CLK----RC1	GROUND--10	GROUND--11	GROUND--12
11	GROUND--13	GROUND--14	GROUND--15	GROUND--16	GROUND--17	GROUND--18	GROUND--19	GROUND--20	GROUND--21	GROUND--22
21	GROUND--23	GROUND--24	GROUND--25	GROUND--26	GROUND--27	GROUND--28	GROUND--29	GROUND--30	GROUND--31	GROUND--32
31	GROUND---1	GROUND---2	GROUND---3	GROUND---4	GROUND---5	GROUND---6	GROUND---7	GROUND---8	GROUND---9	TECTRREAC1
41	TECTRREAC3	TECTRREAC5	T-ALT---F0	T-ALT---H1	T-ALT---F0	T-ALT---F1	T-BINDATA1	T-BIT35-C1	T-BITEC-C1	T-BITE0-L0
51	T-BITE0-L1	T-BITECSC1	T-BITE1-C1	T-BITE1-L0	T-BITE1-L1	T-BITE1SG1	T-BITE2-G1	T-BITE2-L0	T-BITE2-L1	T-BITE2SG1
61	T-BITE3-G1	T-BITE3-LC	T-BITE3-L1	T-BITE3SC1	T-BITE4-C1	T-BITE4-L0	T-BITE4-L1	T-BITE4SC1	T-BITE5-C1	T-BITE5-L0
71	T-BITE5-L1	T-BITE5SC1	T-BITE6-C1	T-BITE6-L0	T-BITE6-L1	T-BITE6SG1	T-BITE7SG1	T-BIT(3-C1	T-BIT-F1C1	T-BIT-I1C1
81	T-BIT--1C1	T-BIT--KCC	T-BIT--KC1	T-BIT--KC2	T-BIT--KG3	T-CARRY1L1	T-CARRY1G1	T-CARRY1I1	T-CARRY2C1	T-CARRY2C1
91	T-CARRY2I1	T-CLK0C--1	T-CLK01--1	T-CLK02--1	T-CLK03--1	T-CLK04--1	T-CLK05--1	T-CLK06--1	T-CLK07--1	T-DSEDATA1
101	T-DSEDPCG0	T-DSEDPCC1	T-DSEDPEC1	T-DSEDPKC0	T-DSEDPKC1	T-DSPDP-C1	T-DSPDP-L3	T-DSFDPSG1	T-FMR47-C1	T-FMR48-L1
111	T-FMR49-C1	T-FMR50-C1	T-FMR51-C1	T-FMR52-G1	T-FMR53-G1	T-FMR54-G1	T-FMR55-G1	T-FMR56-C1	T-FMR57-C1	T-FMR58-G1
121	T-FMR59-C1	T-FMR60-L1	T-FMR63-C1	T-FMTR1RC0	T-FMTR1RG1	T-FMTR1-L0	T-FMTR1-L1	T-FLKGD-C1	T-INIT--FC	T-INIT--F1
131	T-INSFTEG1	T-LED-1-F1	T-LED-2-F1	T-LED---G1	T-LOAD1-H1	T-LOAD2-F1	T-LOAD--G0	T-LOAL--C1	T-MSHF--C5	T-MSHE--C6
141	T-MSHE--C7	T-MSH---C0	T-MSH---L1	T-MSH---D2	T-MSH---D3	T-MSH---D4	T-MSH---D5	T-MSH---C6	T-MSH---C7	T-NULL1--1
151	T-PC1DATA1	T-PARHTCG0	T-PARHTCC1	T-PARHTCC2	T-PARHTKG0	T-PARHTKG1	T-PARHTRG1	T-PARHT-L0	T-PARHT-F1	T-PARHT-LC
161	T-PARHT-L1	T-PARHT-L2	T-PARHTSC1	T-PAREHCC1	T-PAREPEG1	T-PAREPKL0	T-PAREPKG1	T-PAREP--1	T-PAREP-L1	T-PAREHSL1
171	T-NULLT--1	T-RA--F-G1	T-RA-----C0	T-RA----C1	T-RA-----L0	T-RA-----11	T-RA---SG1	T-RD-DF-G1	T-RD--1-LC	T-RD---FC1
181	T-RC----KC0	T-RC----C1	T-RC-----11	T-RC-1--F1	T-RC-2--F1	T-RC-----G1	T-RS-----G1	T-SEED0F-1	T-SEEDG-L1	T-SEEDOSC1
191	T-SEEDDF-G1	T-SEEDFSC1	T-SEEDFL-C1	T-SEEDFKG1	T-SEEDD-KG0	T-SDDD-KG1	T-SDDD--L0	T-SDDD--L1	T-SDDD-SG0	T-SDDD-SG1
201	T-SELVA-G1	T-SELVF-C1	T-SELVC-C1	T-SELVD-G1	T-SELV--G1	T-TA--FKG1	T-TA---KG0	T-TA----G1	T-TA-----C0	T-TA-----L1
211	T-TA---SG1	T-TFG01FC1	T-TFG01KC0	T-TFG01KC1	T-TFG01-L0	T-TRG01-F1	T-TBG01-L3	T-TBG01-G3	T-TBG01-LC	T-TFG01-L1
221	T-TFG02FG1	T-TFG02KG0	T-TFG02KC1	T-TFG02-L0	T-TFG02-L1	T-TRG0FFG1	T-TBG1KEG1	T-TFG---FC	T-TFG---F1	T-TDR00-C1
231	T-TDR00-G1	T-TDR00-L1	T-TDR01-L1	T-TDR01-C1	T-TDR01-L1	T-TDR02-L1	T-TDR02-G1	T-TDR02-L1	T-TDR03-L1	T-TDR03-G1
241	T-TDR03-L1	T-TDR04-C1	T-TDR04-C1	T-TDR04-L1	T-TDR05-L1	T-TDR05-G1	T-TDR05-L1	T-TDR06-C1	T-TDR06-G1	T-TDR06-L1
251	T-TDR07-D1	T-TDR07-C1	T-TDR07-L1	T-TDR--KC0	T-TDR--KG1	T-TDR--KG2	T-TDR--KG3	T-TD-1--F1	T-TD-2--F1	T-TD-----L0
261	T-TD-----G1	T-TFC00-C1	T-TFC00-C1	T-TFC00-L1	T-TFC01-C1	T-TFC01-G1	T-TFC01-L1	T-TFC02-C1	T-TFC02-G1	T-TFC02-C1
271	T-TFC03-C1	T-TFC03-G1	T-TFC03-L1	T-TFC04-C1	T-TFC04-G1	T-TFC04-L1	T-TFC05-G1	T-TFC05-G1	T-TFC05-L1	T-TFC--KC0
281	T-TFC--KG1	T-TFC--KG2	T-TFC--KC3	T-TFRO0FC0	T-TFRO0KG0	T-TFRO0KG1	T-TFROO-L0	T-TFROO-GC	T-TFROO-LC	T-TFROO-L1

```
291   T-TFR00SG1  T-TFR01F00  T-TFR01KG0  T-TFR01KG1  T-TFR01-D0  T-TFR01-D1  T-TFR01-G0  T-TFR01-L0  T-TFR01-L1  T-TFR01SG1
301   T-TFR02ED0  T-TFR02KG0  T-TFR02KG1  T-TFR02-D0  T-TFR02-D1  T-TFR02-G0  T-TFR02-L0  T-TFR02-L1  T-TFR02SG1  T-TFR03FD0
311   T-TFR03KG0  T-TFR03KG1  T-TFR03-D0  T-TFR03-D1  T-TFR03-G0  T-TFR03-G1  T-TFR03-G2  T-TFR03-L0  T-TFR03-L1  T-TFR03SG1
321   T-TFR04ED0  T-TFR04KG0  T-TFR04KG1  T-TFR04-D0  T-TFR04-D1  T-TFR04-G0  T-TFR04-G1  T-TFR04-G2  T-TFR04-L0  T-TFR04-L1
331   T-TFR04SG1  T-TFR05FD0  T-TFR05KG0  T-TFR05KG1  T-TFR05-D0  T-TFR05-D1  T-TFR05-G0  T-TFR05-G1  T-TFR05-G2  T-TFR05-L0
341   T-TFR05-L1  T-TFR05SG1  T-TFR0C-G1  T-TFR0K-G1  T-TFR1C-G1  T-TFR1K-G1  T-TFR2K-G1  T-TFR3C-G1  T-TFR3K-G1  T-TFR4C-G1
351   T-TFR4K-G1  T-TFR5K-G1  T-TFR64-G1  T-TFR--IC1  T-TFR--1G3  T-TFR--IG5  T-TIS25-G1  T-TIS---G1  T-TS----G1  T-VIEW--R0
361   T-VIEW--R1  T-VUINITG0  T-VU-INIT1
```

NUMBER OF INTERNAL SIGNALS= 357

STORAGE ELEMENT SIGNAL LIST CU ADDED=TOISP

T-TFC05-L1	0	T-TFC04-L1	1	A002
T-TFC03-L1	2	T-TFC02-L1	3	A003
T-TFC01-L1	4	T-TFC00-L1	5	A004
T-TDR07-L1	6	T-TDR06-L1	7	A005
T-DSPDP-L1	8	T-DSPDP-L3	9	A016
T-TDR05-L1	10	T-TDR04-L1	11	A019
T-TDR03-L1	12	T-TDR02-L1	13	A020
T-TDR01-L1	14	T-TDR00-L1	15	A021
T-CARRY1L1	16	T-CARRY2L1	17	A022
T-EMTPI-L1	18		******	A032
T-TFR05-L1	19	T-TFR05-G1	20	A035
T-TFR04-L1	21	T-TFR04-G1	22	A040
T-TFR03-L1	23	T-TFR03-G1	24	A045
T-TFR02-L1	25	T-TFR02-G0	26	A050
T-TFR01-L1	27	T-TFR01-G0	28	A055
T-TFR00-L1	29	T-TFR00-G0	30	A060
T-RITE0-L1	31	T-RITE1-L1	32	A085
T-RITE2-L1	33	T-RITE3-L1	34	A086
T-RITE4-L1	35	T-RITE5-L1	36	A087
T-RITE6-L1	37	T-RITE6-L2	38	A089
T-TPG01-L1	39		******	A101
T-TPG02-L1	40		******	A104
T-PARRT-L1	41	T-PARRT-L2	42	A108
T-PARER-L1	43		******	A112
T-RA----L1	44		******	A116
T-TA----L1	45		******	A120
T-RD----L1	46		******	A124
T-SDDD--L1	47		******	A135

T-TFC05-L1	15872
T-TFC04-L1	15873
T-TFC03-L1	15874
T-TFC02-L1	15875
T-TFC01-L1	15876
T-TFC00-L1	15877
T-TDR07-L1	15878
T-TDR06-L1	15879
T-DSPDP-L1	15880
T-DSPDP-L3	15881
T-TDR05-L1	15882
T-TDR04-L1	15883
T-TDR03-L1	15884
T-TDR02-L1	15885
T-TDR01-L1	15886

APPENDIX H

A SAMPLE OUTPUT LISTING OF <BOARD NAME>/SIMULA

```
                    ILLIAC IV CONTROL UNIT CARD LOGIC SIMULATOR

                          TOFUND SIMULATOR VERSION  1

SIMULATION NUMBER   9                    MONDAY,  1/05/70,  9:49 PM


SIMULATOR HISTORY:

          PROGRAM              DATE AND TIME RUN          RUN #        LATEST

          NETLIST          8/08/69,  12:00 AM              1
       CUBOARD/UPDATE     10/17/69,   2:23 PM              1
       CUBOARD/WEDTR1     10/17/69,   2:24 PM              1
       CUBOARD/WEDTR2     10/17/69,   2:26 PM              1
       CUBOARD/LEVELER    10/17/69,   2:27 PM              1
       CUBOARD/REDUCE     10/17/69,   2:27 PM              1
       CUBOARD/HSKEEP     10/17/69,   2:28 PM              1
       CUBOARD/SIMGEN     10/17/69,   2:28 PM              1


SIMULATION CONTROLLED BY  IFSLA PROGRAM NUMBER  10 CREATED 11/09/69,  11:55 AM
```

S I M U L A T I O N L O G

STEP NUMBER	STEP LABELED
1	STEP01
2	STEP02
3	STEP03
4	STEP04
5	STEP05
6	STEP06
7	STEP07
8	STEP08
9	STEP09
10	STEP10
11	STEP11
12	STEP12
13	STEP13
14	STEP14
15	STEP15
16	STEP16

```
CPU TIME USED =  0 MIN  3 SEC
I/O TIME USED =  0 MIN 13 SEC
TOTAL TIME USED =  0 MIN 12 SEC

16 STEPS SIMULATED AT  76 STEPS PER MINUTE
```

APPENDIX I

A PORTION OF AN OUTPUT LISTING FROM CUBD/PRINTER

SIMULATION STEP NUMBER 10 -- STEP LABELED "STEP1"

```
10.01:
        COMP:          ACTUAL  00000000 00000000 000 (2)
        FF:            ACTUAL  00000000 00000000  (2)
        TRU:           ACTUAL  00000000 11111111  (2)
        TGC:           ACTUAL  00000000  (2)
10.02:
        COMP:          ACTUAL  00000000 00001111 000 (2)
        FF:            ACTUAL  00000000 00000000  (2)
        TRU:           ACTUAL  00000000 11111111  (2)
        TGC:           ACTUAL  00000000  (2)
10.03:
        COMP:          ACTUAL  00000000 11110000 000 (2)
        FF:            ACTUAL  00000000 00000000  (2)
        TRU:           ACTUAL  00000000 11111111  (2)
        TGC:           ACTUAL  00000000  (2)
10.04:
        COMP:          ACTUAL  00000000 11111111 010 (2)
        FF:            ACTUAL  00000000 00000000  (2)
        TRU:           ACTUAL  00000000 11111111  (2)
        TGC:           ACTUAL  00000000  (2)
10.05:
        COMP:          ACTUAL  00001111 00000000 000 (2)
        FF:            ACTUAL  00001111 00000000  (2)
        TRU:           ACTUAL  00001111 11111111  (2)
        TGC:           ACTUAL  00000000  (2)
10.06:
        COMP:          ACTUAL  00001111 00001111 000 (2)
        FF:            ACTUAL  00001111 00000000  (2)
        TRU:           ACTUAL  00001111 11111111  (2)
```

SIMULATION STEP NUMBER 2 -- STEP LABELED "STEP2"

```
2.01:
      TRD:                    ACTUAL  11111111 11111111  (2)
      COMP:                   ACTUAL  00000000 00000000 000 (2)
      TGC:                    ACTUAL  00000000  (2)
      FF:                     ACTUAL  11111111 00000000  (2)
2.02:
      TRD:                    ACTUAL  11111111 11111111  (2)
      COMP:                   ACTUAL  00000000 00001111 000 (2)
      TGC:                    ACTUAL  00000000  (2)
      FF:                     ACTUAL  11111111 00000000  (2)
2.03:
      TRD:                    ACTUAL  11111111 11111111  (2)
      COMP:                   ACTUAL  00000000 11110000 000 (2)
      TGC:                    ACTUAL  00000000  (2)
      FF:                     ACTUAL  11111111 00000000  (2)
2.04:
      TRD:                    ACTUAL  11111111 11111111  (2)
      COMP:                   ACTUAL  00000000 11111111 010 (2)
      TGC:                    ACTUAL  00000000  (2)
      FF:                     ACTUAL  11111111 00000000  (2)
2.05:
      TRD:                    ACTUAL  00000000 00000000  (2)
      COMP:                   ACTUAL  00001111 00000000 000 (2)
      TGC:                    ACTUAL  00000000  (2)
      FF:                     ACTUAL  11111111 00000000  (2)
2.06:
      TRD:                    ACTUAL  00000000 00000000  (2)
      COMP:                   ACTUAL  00001111 00001111 000 (2)
      TGC:                    ACTUAL  00000000  (2)
```

LIST OF REFERENCES

[1] Abel, L.C., Tanaka, C. "Proposal for Control Unit Board Logic Simulator System." Diagnostic Group Memorandum (DM-031). Department of Computer Science, University of Illinois, November 1968.

[2] Allegre, N.G. "TESLA, A control Language for Logic Simulations of Digital Circuits." Report No. 347. Department of Computer Science, University of Illinois, August 1969.

[3] Berge, C. "Theory of Graphs and Its Applications." John Wiley & Sons, Inc. 1962.

[4] Carroll, A.B., Kato, M., Koga, Y., Naemura, K. "A Method of Diagnostic Test Generation." SJCC 1969, pp 221-228.

[5] Harary, F., Norman, R.Z., Cartwright, D. "Structural Models: An Introduction to the Theory of Directed Graphs." John Wiley & Sons, Inc. 1965.

[6] Kaufman, A. "Graphs, Dynamic Programming, and Finite Games." Academic Press. 1967.

[7] Ramamoorthy, C.V. "Analysis of Graphs by Connectivity Considerations." JACM Vol. 13, No. 2, April 1966, pp 211-222.

[8] Tanaka, C. "Level Assignment Problem for Test Simulator Generator," Diagnostic Group Memorandum (DM-035). Department of Computer Science, University of Illinois, October 1968.

[9] Abel, L.C., Naemura, K., Tanaka, C. "Parallel Logic Simulator and its Use for Test Generation." Workshop on Reliability and Maintainability of Computer Systems at Lake of the Ozarks. October 1969.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

PARALLEL SIMULATION OF DIGITAL SYSTEMS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

5. AUTHOR(S) (First name, middle initial, last name)

Chiyozi Tanaka

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 30, 1970	99	9

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
USAF 30(602)-4144	DCS Report No. 382
b. PROJECT NO. 46-26-15-305	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	ILLIAC IV Document No. 211

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies from DCS.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

13. ABSTRACT

The paper describes the method used to generate the parallel simulator developed for the ILLIAC IV Project, University of Illinois. The basic approach to achieve an efficient simulator system is parallel logic simulation, package-level simulation, the use of a high level language (ALGOL) and a flexible simulation control language (TESLA). This paper analyzes the problems associated with the ideas and gives the algorithms used to generate the simulator. Emphasis is placed on the algorithms and the implementation of the simulator generator system.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
logic design						
logic simulator						
design automation						
diagnostics						
diagnostic generation						
ILLIAC IV						